计算机基础教育丛书

# COBOL 语 言（下册）

修订版

谭浩强 编著

清华大学出版社

# 目　录

# 第七章　数据部之二

## ——数据部的较高技巧

## §7.1　数据在计算机内的表示形式

至今为止，我们介绍的是数据在计算机内部存放的最简单的情况，即：一个字符在计算机内存中占一个字节，无论是数字（0～9）或字母（A～Z）或其它字符（如：＋，$，＊…）都各占一个字节。在本章我们将介绍其它的形式。

### 7.1.1　计算机内存的组织形式

在计算机中是以二进制形式来表示数据的。内存单元是由一系列的二进制位（它的值是0或1）组成的，因此它的最小单位是二进制的“位”（bit）。

若干位组成一个字节（byte），在大多数机器中一个字节为8个二进位。在计算机高级语言中一般是以字节来作最小处理单位的，即存取数据以字节为单位。譬如说，取数据时一次至少要取出一个字节8个二进位的内容，如00100100等。

字节又可组成半字、字、双字。各种不同计算机对一个字包含几个字节有不同的规定。一般计算机以4个字节（32位）作为一个“字”（word），以2个字节（16位）作为“半字”，以8个字节（64位）作为一个“双字”。

数据按指定形式存放在计算机的内存中。如一个字符可以用ASCII代码或EBCDIC代码表示（见附录Ⅶ），把它放到计算机内存中，需要占一个字节（8位二进制）。从内存中取数据时，则是取出这个字节内容。

### 7.1.2　字符数据在内存中的存储形式

字符型、字母型和数值编辑型、字符编辑型数据项中的数据，每一个字符都在内存中占一个字节。这种形式称为标准数据形式。

由于内存中数据都是以二进制数来表示的，因此要规定每一个字符用怎样的一组二进制数来表示。每类计算机系统分别选择其所用的代码形式。例如，在ASCII代码中，以8位二进制数01000001代表“A”，以00110001代表“1”。而在EBCDIC代码中，以11000001代表“A”，以11110001代表“1”。总之，一个字符以8位二进制数代表，占一个字节。字符转换成二进制代码不是由用户自己转换的，而是由计算机系统自动转换的。例如在键盘上按下A，B，C三个字符，就把相应的二进制代码输入到计算机内存中去。反之，从计算机中输出一个数据，先将这些字节中存放的二进制数（如ASCII码的01000001，01000010，01000011）取出，再把它们转换成字符（如A，B，C）打印出来。

如果采用字符型数据形式，不论是字母或数字，都按一个字节存放一个字符。如有两个数据项A和B，它们的描述为：

77 A PIC X (3) VALUE'ABC'.

77 B PIC X (3) VALUE'123'.

在计算机内存中的实际存储情况如下表所示：数据项 A 占三个字节，如果计算机系统用的是 ASCII 码，则第一个字节中的数据是 01000001，如果采用 EBCDIC 码，则第一个字节中为 11000001，其余类推。

| | A | | | |
|---|---|---|---|---|
| 实际的二进码 | 01000001<br>(11000001) | 01000010<br>(11000010) | 01000011<br>(11000011) | ASCII 码<br>EBCDIC 码 |
| 代表字符 | 'A' | 'B' | 'C' | |

| | B | | | |
|---|---|---|---|---|
| 实际的二进码 | 00110001<br>(11110001) | 00110010<br>(11110010) | 00110011<br>(11110011) | ASCII 码<br>EBCDIC 码 |
| 代表字符 | '1' | '2' | '3' | |

特别需要指出的是，如果数据描述形式为字符型（例如本例中描述为 X (3)），则非数值常量中的数字亦按一般字符处理，如上述“1 2 3”，仅意味三个字符 1，2，3，而不代表一百二十三。

### 7.1.3 数值型数据在内存中的存储形式

数值型数据在内存中存放的形式有以下几种：

(一) 外部十进制（或称扩张十进制）形式

按数值在机器外部的表现形式，一个数字在内存中占一个字节。如：

77 C PIC 9 (3) VALUE 486.

在内存中占三个字节。每一个数字与二进制代码的关系同上述。在内存中实际内容如下（假设机器内用 EBCDIC 码）。

| C | | |
|---|---|---|
| 11110100 | 11111000 | 11110110 |
| '4' | '8' | '6' |

为表示方便，有时用十六进制数来表示一个数。一个十六进制数包括四个二进位数，用十六进制的“0”代表二进制数 0000，“1”代表 0001，“2”代表 0010，“3”代表 0011，“4”代表 0100，“5”代表 0101，“6”代表 0110，“7”代表 0111，“8”代表 1000，“9”代表 1001，“A”代表 1010（十进制的 10），“B”代表 1011（十进制的 11），“C”代表 1100，“D”代表 1101，“E”代表 1110，“F”代表 1111（F 就是十进制中的 15）。

因此 C 在内存中的情况可以表示为：

| F4 | F8 | F6 |
|---|---|---|
| '4' | '8' | '6' |

如果数据项 D 的描述如下（D 的值为负）：

77 D PIC S9 (3) VALUE -486.

此时，负号不占一个字节，而在最后一个字节中放入某个信息，一般是将此字节的前四位 1111 改为 1101，后四位仍为 0110。即

| D | | |
|---|---|---|
| 11110100 | 11111000 | 11010110 |

计算机检查最后一个字节的前四位，如为 1101，则按负数处理，如为 1100，按正数处理。或者说，用十六进制中“C”（1100）代表“正”，“D”（1101）代表“负”，“F”（1111）代表无符号，即绝对值（也有些计算机系统不用“C”，“D”，而用其它数代表“正”，“负”）。

（二）外部浮点数形式

某些数据，它的值很大或很小（如 $+1.23876\times10^{59}$ 或 $-1.38457\times10^{-69}$），用以前讲的外部十进制形式存储是有困难的。COBOL 允许用指数形式来表示一个数，例如上面两个数，用指数形式可以写为：

+1.23876E+59

−1.38457E−69

其中“E”表示“以 10 为底的指数”。“E”前面的部分称为“数值部分”或“尾数部分”，“E”后面的是“指数部分”或“阶码部分”。数值部分和指数部分各有一个符号以表示正或负。其一般形式为：

**数符 数值部分 E 阶码符 阶码**

为了表示这种指数形式的数据（外部浮点形式），在 PIC 子句中可以这样写：

77 A PIC +9.99999E+99.

或 77 B PIC +9V99999E−99.

它表示在内存中按以上形式存放数据。其中每一个“9”表示此位置可放入一个 0～9 间的数字，一个“9”占一字节。E 前面放数值部分，E 后面放指数部分。小数点“.”表示此位置有一小数点，“V”表示此位置有一隐含的小数点，它不占内存字节。正号“+”的意思是，如果数值为正，此处为正号，数值为负时，此处为负号。负号“−”的意思是：数值为正时此处空白，数值为负时，此处为负号（它们的用法和数值编辑项中的“+”、“−”编辑符相同）。“E”也占一个字节。因此，A 在内存占 12 个字节，B 占 11 个字节。如果 A 的值为 $1.23876\times10^{59}$，B 的值为 −1.38457E69，则 A 和 B 在内存中的情况为

A：+1.23876E+59（12 字节）　　B：−1V38457E 69（11 字节）

在多数 COBOL 版本中浮点数可以表示的数的范围是 $5.4\times10^{-79}$ 到 $0.72\times10^{76}$。用外

部浮点形式，PIC 字符串中数值部分最多可以出现 16 个“9”，指数部分除“E”外，应有一个“+”或“−”，以及两个“9”（不应有三个或更多个“9”，因为不可能出现 $10^{100}$ 以上的数值）。

对外部浮点项不能用 VALUE 子句赋初值，如下面写法是不对的：

```
02  N  PIC  +99.99E+99  VALUE  +12.45E+45.
```

只能从外部文件上读入以指数形式表示的数据（例如可以从卡片或磁盘上相应的位置开始，按列依次输入“+12.45E+45”），也可以用 MOVE 语句给 N 传送数值。

下面是一些例子：

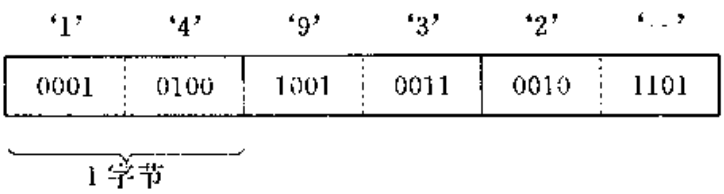
| PIC 字符串 | 外部数据格式 | 表示的值 |
|---|---|---|
| +99V9E−99 | ␣183E−13 | $+18.3\times10^{-13}$ |
| −9V99E+99 | ␣687E+65 | $+6.87\times10^{65}$ |
| +9(3).99E+99 | +875.46E 12 | $+875.46\times10^{-12}$ |
| V9(6)E+99 | ␣678123E+37 | $+0.678123\times10^{37}$ |
| +.99E−99 | +.87E␣62 | $+0.87\times10^{62}$ |

（三）内部十进制（又称缩合十进制）形式

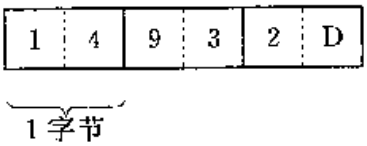
外部十进制形式在内存中一个字节放一个字符。数值型数据只用到 0～9 十个数字，如 129，23868 等。我们从下表中可以看到，0～9 的代码前四位是相同的。

| 十进制数字 | EBCDIC 码 | ASCII 码 |
|---|---|---|
| 0 | 1111 0000 | 0011 0000 |
| 1 | 1111 0001 | 0011 0001 |
| 2 | 1111 0010 | 0011 0010 |
| 3 | 1111 0011 | 0011 0011 |
| 4 | 1111 0100 | 0011 0100 |
| 5 | 1111 0101 | 0011 0101 |
| 6 | 1111 0110 | 0011 0110 |
| 7 | 1111 0111 | 0011 0111 |
| 8 | 1111 1000 | 0011 1000 |
| 9 | 1111 1001 | 0011 1001 |

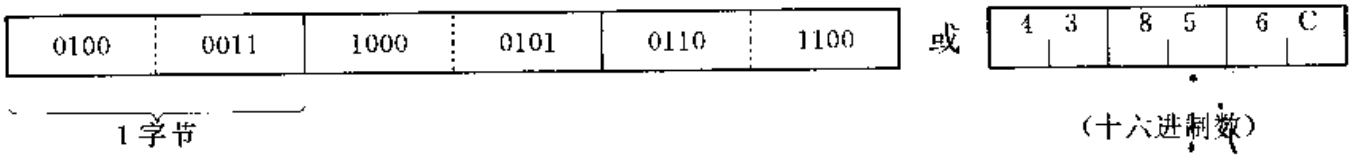
因此在数值型数据的前提下，为表示 0～9，前四位并不起辨别作用，只是根据后四位的不同来判别是 0～9 中的哪个数字。因此，为节省内存，可以只用四位二进制数字来代表一个十进制数。在一个字节中放两个十进制数字。每个数字占四个二进位，即半个字节。符号也占半个字节。如，−14932 在内存中为：



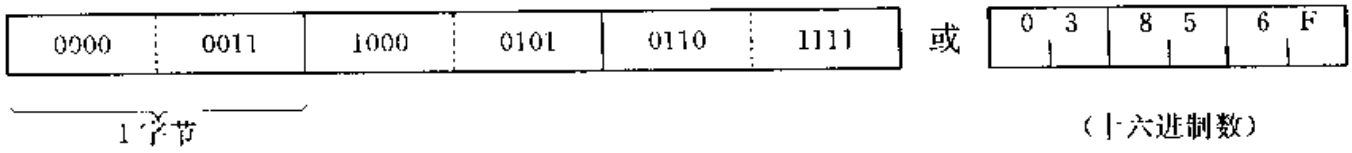
或用十六进制表示（十六进数的一位代表四个二进位）：




这种存放形式称内部十进制，或“缩合十进制”，+43856 在内存中为：




无符号的数 3856，在内存中为：




F 表示数值无符号，即取绝对值。由于存取的最小单位是字节，因此 3856 虽然只需两个半字节，但也占二个字节，最前面半个字节补零。

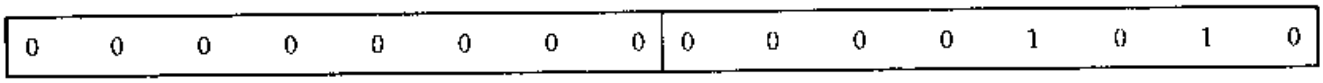
（四）定点二进制形式

不是以一个数字对应一个字节或半个字节，而是先把十进制数化成定点二进制数形式，然后存放在内存中。如：10 先化成 $(1010)_2$，括弧外下角 2 表示是二进制的数。然后把 1010 放到内存单元中。

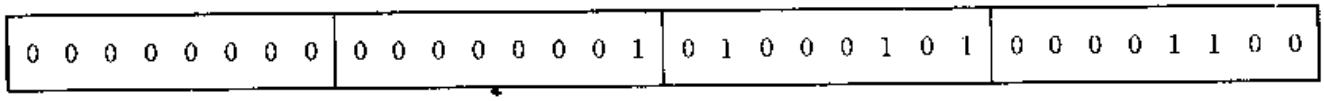
如果在数据部中以 PIC 9（4）描述一个数据项，并指定为定点二进制形式，数据长度为 4 位数字，则需要在内存中开辟二个字节。COBOL 规定在内存中根据数据项的长度分别用二字节、四字节或八字节来存放一个以定点二进制形式存放的数。

| 在 PIC 子句中描述字符 ‘9’ 的个数 | 占内存字节 |
|---|---|
| 1～4 | 2 |
| 5～9 | 4 |
| 10～18 | 8 |

例如，十进制数 10，如果以 PIC 9（2）描述，在内存中以定点二进制形式存放，则占两个字节：




十进制数 $(83212)_{10}$用二进制形式表示为 10100010100001100，由于用 PIC 9（5）在内存中占四个字节，存放形式为：




因为 2 个字节放不下十进制中所有的 5 位整数（2 个字节存放数的范围为－32767～

32768)，所以系统规定5位以上整数用4个字节存放。同样，4个字节放不下所有的10位整数，故规定10位以上的整数一律以8个字节存放。

（五）内部浮点形式

它以内部的指数形式（二进制的指数形式）来表示一个数，以固定长度的内存单元来存放一个数。如以4个字节（32位）表示一个数，其中8位表示指数部分，24位表示数字部分。这称为短浮点形式。


也可以用8个字节（64位）表示一个数，指数部分仍为8位，数字部分为56位，称长浮点形式。长浮点形式比短浮点形式有更高的精确度（内部浮点形式的数值在内存中所占字节长度只有4个字节和8个字节两种）。

无论短浮点形式或长浮点形式，所表示的数值范围均为：

$$5.4\times10^{-79}\sim0.72\times10^{+6}$$

### 7.1.4 数据描述与存储形式的关系

（一）字母型、字符型、编辑型、外部十进制数据和以外部浮点形式表示的数据用标准数据形式来存放，即一个字符占一个字节。

```
如：77  A   PIC  X (6).
    77  B   PIC  A (6).
    77  C   PIC  999.99.
```

均占6个字节。

（二）数值型数据可以由程序员任意选定存放形式（外部十进制、外部浮点形式、内部十进制、定点二进制、内部浮点形式）。用外部十进制形式最简单、最好理解，但一般说，它占内存多。如135879326九位数，用外部十进制需9个字节，用内部十进制需5个字节，用定点二进制需4个字节、短浮点形式需4个字节，长浮点形式需8个字节。

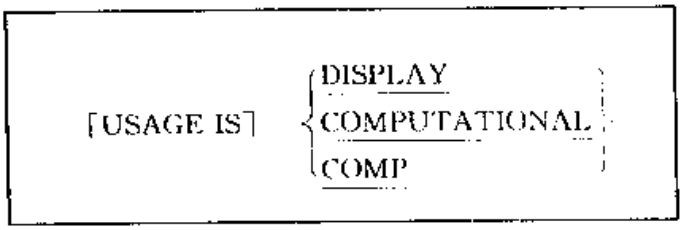
（三）数据在计算机内进行运算，都是化成二进制数以后再进行的，因此，外部十进制形式是不能直接进行运算的，需要由计算机把它们先化为定点二进制或内部浮点形式，然后再进行运算。假定A和B已定义为外部十进制形式，其值分别为11和12，它在内存中的存储形式为F1|F1和F1|F2（假设以EBCDIC代码存放），A和B各占两个字节。如果执行ADD A TO B，即A+B→B，并不是将A和B中各自第一个字节“F1”相加，第二字节“F1”和“F2”相加。而是先将A和B化成$(1011)_2$和$(1100)_2$，然后进行二进位的相加，相加后得$(10111)_2$，把它转换成23的EBCDIC码，即“F2”和“F3”，再存到B中。计算前后各要转换一次。显然，花时间多，速度慢。

从运算速度上看，定点形式最快，内部浮点形式次之，外部浮点形式和外部十进制、内部十进制最慢。因此，用户应根据需要选择用哪一种数据形式。这就要用下一节介绍的USAGE子句。

## §7.2 用法子句（USAGE 子句）

使用用法(USAGE)子句可以使程序设计者自由选择数据在内存中的存放形式。譬如，有数据项 A 和 B，需要多次对它们进行运算，如果用外部十进制形式则来回转换会大大降低运算速度，这时，可以选择 A 和 B 为定点二进制形式或内部浮点形式。如果数据项 C 和 D 参加运算次数少，而且需要多次打印出 C 和 D 的结果，这时用外部十进制比较适合，因为它最适合打印的要求，不必再进行转换。

USAGE 子句的一般格式为：




其中“USAGE IS DISPLAY”的意思是“显示型的用法”，即：此数据项适于显示、打印，它采用标准数据形式（一个字节放一个字符）。

COMPUTATIONAL 和 COMP 是同一意思（COMP 是 COMPUTATIONAL 的缩写）。“USAGE IS COMP”的意思是“计算型的用法”（它只能用于数值型数据项）。表示此数据形式适于计算，它采用适于计算用的定点二进制形式或内部浮点形式。

标准 COBOL 只列出 DISPLAY 和 COMPUTATIONAL (COMP) 两种用法的格式。各种计算机还规定了它可采用的有哪几种数据存放形式以及用什么来代表它们。例如，在 M 系列，IBM 系列和许多计算机系统中提供的功能包含：

| | |
|---|---|
| DISPLAY | （标准数据形式，一个字节放一字符） |
| COMPUTATIONAL<br>COMP | （定点二进制形式） |
| COMPUTATIONAL－1<br>COMP－1 | （内部短浮点形式） |
| COMPUTATIONAL－2<br>COMP－2 | （内部长浮点形式） |
| COMPUTATIONAL－3<br>COMP－3 | （内部十进制形式） |

在使用前，应查阅所用计算机的 COBOL 说明书。

说明：

（一）USAGE 子句是用来指定数据项在内存中的存储形式的。如：

```
02  A  PIC  9 (4)  USAGE  IS  COMP.
```

表示数据项 A 用定点二进制形式（假设该计算机系统所用的 COBOL 按我们介绍的上述规定）。

USAGE IS 这两个英文字可以省写。上面的数据项 A 描述体可简写成：

```
02  A  PIC  9 (4)  COMP.
```

（二）如省略 USAGE 子句，则隐含表示为用 DISPLAY 形式。

```
如  02  B  PIC  9 (6)  USAGE IS DISPLAY.
```

```
02  B  PIC  9(6)  DISPLAY.
02  B  PIC  9(6).
```

以上三种写法等价。字符型、字母型、编辑型、外部十进制、外部浮点形式的数据项必须用 USAGE DISPLAY（可以采用显式或隐含形式表示）。

（三）如果对组合项描述为某一种存储形式，则表示这个组合项的下属各初等项都是这种形式。如：

```
01  T.
    02  T1  USAGE  COMP.
        03  X  PIC  S9(3).
        03  Y  PIC  S9(3).
```

在 T1 的描述中用了 COMP，表示定点二进制，它下属的 X 和 Y 也都是定点二进制形式。它相当于：

```
01  T.
    02  T1.
        03  X  PIC  S9(3)  COMP.
        03  Y  PIC  S9(3)  COMP.
```

组合项的描述中如果用了用法子句，其下属的初等项可以再用用法子句，但二者的说明不应矛盾。例如将上面的 02 层 T1 描述体改写为：

```
02  T1  USAGE  COMP.
```

是可以的，因它与 03 层中用的一致。但如果改成：

```
02  T1  DISPLAY.
```

则 02 层与 03 层描述发生矛盾。不允许。

（四）USAGE 子句指定的数据存储形式不应与 PIC 子句指定的数据类型矛盾。如：

```
    04  A  PIC  A(4)      USAGE  COMP.
或  04  B  PIC  $99.99    COMP  1.
```

都是错误的。前者 A 是字母型数据，不能采用数值型的存储形式。后者 B 是编辑数据型数据，当然也不能采用短浮点形式。

（五）长、短浮点形式已确定了内存的长度，不应再用 PIC 子句。如：

```
04  A1    COMP--2.
```

没用 PIC 子句，但已确定为长浮点形式，如果某一具体计算机规定长浮点形式占 8 个字节，则 A1 不论是什么值，在内存中都占 8 个字节。

（六）在传送或运算时不同存储形式的数值型数据间可互相转换。

```
如：77  A  PIC  9(3)  COMP.
    77  B  PIC  999V99  DISPLAY.
```

二者的存储形式不同。如果有语句：

```
MOVE  A  TO  B.
```

则先将 A 转换成 B 的形式再存入 B。又如：

```
ADD  A  TO  B
```

也由编译系统进行自动转换，使 A 和 B 具有同一形式，相加后按 B 的形式存入 B。

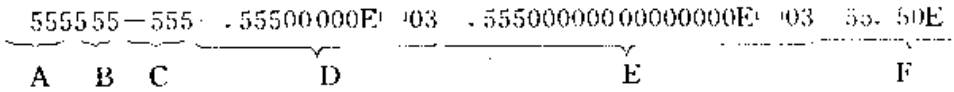
（七）在用 DISPLAY 语句显示数据项的内容时，如果数据项的 USAGE“用法中”不是指定 DISPLAY，则在显示前，计算机自动将内存中数据形式转换成 EBCDIC 码（或 ASCII 码，视计算机系统而定），然后以字符形式显示。例如：

```
01  LP-REC.
    02  FILLER  PIC  X.
    02  A  PIC  S999  COMP.
    02  B  PIC  999  COMP-3.
    02  C  PIC  S999.
    02  D  COMP-1.
    02  E  COMP-2.
    02  F  PIC  +99.99E+99.
```

如果执行 MOVE－555 TO A，B，C，D，E，F.

然后执行 DISPLAY A，B，C，D，E，F.

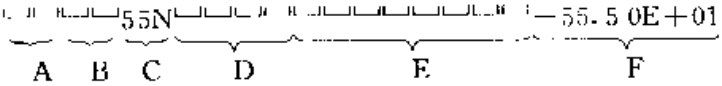
显示出的结果为：




即先转换，后显示。COMP－1（内部短浮点形式）显示时给出 8 位有效数字，COMP－2（内部长浮点形式）显示时给出 17 位有效数字（但不同计算机系统规定不同）。

可以看出，数值型的数据，不论指定的是什么“用法”，都可以进行计算，也都可以用来显示，只是在不同的场合下不同存储形式的效率不同，程序员根据需要用 USING 子句选择较佳的“存储形式”。

（八）如果用 WRITE 语句，则直接输出，不进行转换。例如在上面例子中在 MOVE 语句之后，执行 WRITE LP-REC AFTER 1. 则打印结果为：




由于 A，B，D，E 的内容不是 DISPLAY 用法的，它们无法输出相应的字符，故按字节数留空白，只有 C 和 F 是 DISPLAY 用法的，按内存中存储的实际情况输出相应的字符。C 在内存中的存放情况为：F5 F5 D5，按 EBCDIC 码“F5”相应于字符“5”，“D5”相应于字符“N”，故输出“55N”。

但如果将 A 传送给一个数据项 G，G 是用 PIC X（4）描述的：

```
02  G    PIC  X (4).
```

然后再用 WRITE 语句将 G 所在的记录输出，则在 G 的区域上打印出：555␣。因为在用 MOVE 语句向 G 传送时，将 555 传送给 G（数值的负号按规定不能传送入字符型数据项），左对齐，右补空格。故输出“555␣”。其它数据项（如 B，C，D，E）都与此相同。

关于传送时的规则细节，必要时可以查阅说明书。在此，只要求知道，用 WRITE 语句输出时，是直接按内存中存放数据形式输出（不加转换的）。这是和用 DISPLAY 语句时不同的。

## §7.3 符号子句（SIGN 子句）

SIGN 子句用来指定数值型数据描述体中运算符号的状态和位置。

（一）在没有 SIGN 子句时，数值的符号是存放在数据项最后一个字节中的。如：

```
02  A  PIC  9(3)  USAGE DISPLAY.
```

若 A 的值为“012”，当机器内码为 EBCDIC 码时，则在内存中 A 的值是以下面的形式存放的：

| F0 | F1 | F2 |
|---|---|---|

最后一个字节用十六进制码表示是“F2”，即二进码 11100010。

如果在 PIC 子句中用“S”描述符，如：

```
02  A  PIC  S 9 (3)  USAGE DISPLAY.
```

则以最后一个字节的前半个字节作为符号的标志。如果 A 的值为“+012”，则在内存中 A 的值是这样存放的：

| F0 | F1 | C2 |
|---|---|---|

（二）用 SIGN 子句可以指定符号在数值的前部还是后部。

如果在 A 的描述体中包含以下的 SIGN 子句：

```
02  A  PIC  S 9 (3)  USAGE  DISPLAY  SIGN  IS  LEADING.
```

则表示符号不在最后一个字节中表示，而用第一个字节中前四位表示。如果 A 的值仍为“+012”，则第一个字节中的前四位用来存放符号，用十六进制的 C（1100）代表“+”。

| C0 | F1 | F2 |
|---|---|---|

如 A 的值为“−012”，则第一字节的前四位为 D（1101），即：

| D0 | F1 | F2 |
|---|---|---|

如果 SIGN 子句形式改写为：

```
02  A  PIC  S9 (3)  USAGE  DISPLAY  SIGN  IS  TRAILING.
```

则表示符号在最后字节中表示，和不写这个 SIGN 子句时作用相同。也就是说，当省略 SIGN 子句时，隐含“SIGN IS TRAILING”。

（三）指定符号单独占一个字节，用“SEPARATE”可选项。如：

```
02  A  PIC  S9 (3)  USAGE  IS  DISPLAY  SIGN  IS  TRAILING  SEPARATE.
```

表示符号在数值的后部，符号单独占一个字节。A 值为“+012”时，内存区中情况为：

| F0 | F1 | F2 | 4E |
|---|---|---|---|

最后一个字节用来放“+”号。如果用二进制数表示，为：

| 1111 0000 | 1111 0001 | 1111 0010 | 0100 1110 |
|---|---|---|---|
| ‘0’ | ‘1’ | ‘2’ | “+” |

此时多占一个字节用来放符号标志，在 EBCDIC 码中“+”是十六进制的“4E”，即二

进制的“0100 1110”，“—”是十六进制的“60”，即二进制的“0110 0000”。如果写成：

```
02  A  PIC  S9 (3)  SIGN  IS  LEADING  SEPARATE.
```

则符号单独占一个字节，它的位置在第一个字节。当 A 为“—012”时，内存中情况为：

| 6 0 | F0 | F1 | F2 |
|---|---|---|---|

即：

| 0110 0000 | 1111 0000 | 1111 0001 | 1111 0010 |
|---|---|---|---|
| '-' | '0' | '1' | '2' |

SIGN 子句的一般格式为：

$$\left[\text{SIGN IS}\right]\left\{\begin{matrix}\underline{\text{LEADING}}\\ \underline{\text{TRAILING}}\end{matrix}\right\}\left[\underline{\text{SEPARATE}}\ \text{CHARACTER}\right]$$

在使用 SIGN 子句时应注意：

(1) 它只能用于 PIC 字符串中含有“S”的数值型数据描述体中。

(2) 使用 SIGN 子句的数据项的用法 (USAGE) 应当是 USAGE DISPLAY (显式的或隐含的)。不能用于计算型用法的数据项。例如：

```
77  H  PIC  S 999  COMP  SIGN  LEADING  SEPARATE.
```

是不合法的。

(3) 用 SEPARATE 可选项时，内存中增加了一个字节，用来放符号标志，如上例中数据项 A 用 S9 (3) 描述，实际上占四个字节。

但数值型数值在向字符型数据项传送时，符号不予传送，相当于没有加这个符号一样。如有：

```
77  W  PIC  X (4).
77  A  PIC  S9 (3)  VALUE  -125  SIGN  LEADING  SEPARATE.
```

在数据项 A 中存放四个字符“—125”的代码。

| - | 1 | 2 | 5 |
|---|---|---|---|

在执行 MOVE A TO W 后，由于符号不传送，W 中从左到右存放“125␣”。即

| 1 | 2 | 5 | ␣ |
|---|---|---|---|

(4) 如果一个数据项的描述体中包含 SIGN 子句，则数据项的值应包括正或负的符号，否则会出错。

## §7.4 重定义子句 (REDEFINES 子句)

不同的数据项可以共用内存中的同一段空间。例如已给数据项 A 分配了一段内存空

间，在经过某一段的过程后，A 已不再使用了，但它仍占着内存中这部分空间，为了节约内存，可以将另一数据项 B 也分配在 A 所占的这段内存空间。如

```
02  A   PIC  X (5).
02  B   REDEFINES  A   PIC  9 (5).
```

表示 B 对 A 进行重新定义。不仅与其共用空间，而且可以重新定义类型。A 原为字符型，用 PIC X (5) 描述，而 B 用 PIC 9 (5) 描述，定义为数值型。

重定义子句也可以用于组合项，如：

```
02  A.
  03  A1  PIC  9 (4).
  03  A2  PIC  X (6).      共占 14 字节
  03  A3  PIC  X (4).
02  B   REDEFINES  A.
  03  B1  PIC  X (5).
  03  B2  PIC  9 (6).      共占 11 字节
  03  B3  PIC  99V9.
```

B 对 A 进行重新定义，原来 A 组合项共包含 14 字节，令 B 也占同一段内存，但数据结构形式改变了。见图 7.1。

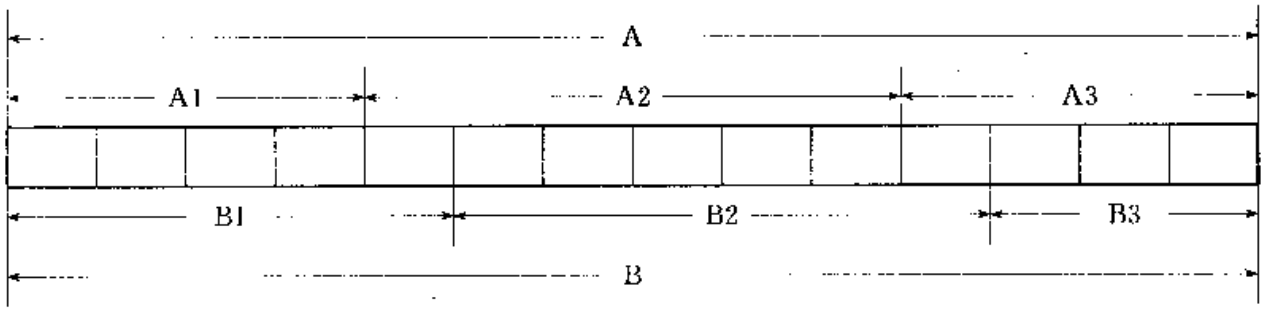



图 7.1

不仅 B1，B2，B3 和 A1，A2，A3 各占的内存字节数不同，而且其类型也不同。

重定义子句的一般格式为：

| 层号 | 数据名 1 | REDEFINES | 数据名 2 |
|---|---|---|---|

表示数据名 1 与数据名 2 共占内存中同一部分区间。

说明：

(一) 数据名 2 与数据名 1 的层号必须相同。即它们应是同一层次的。下面的用法是错误的。

```
02  K   PIC  9 (6).
03  L   REDEFINES  K   PIC  X (6).
```

这样，03 层属于 02 层，与原意不同了。而

```
02  K.
  03  K1  PIC  9 (6).
  03  K2  PIC  X (4).
02  L   REDEFINES  K   PIC  X (10).
```

是正确的。如把最后一行层号改成 03，则错。

REDEFINES 子句不能用于 88 层（定义“条件名”）和 66 层（重命名）。关于“重命名”的说明见下一节 §7.5。

（二）用 REDEFINES 子句的描述体应紧跟在被重新定义的数据项的描述之后，中间不能插入其它项的描述说明。如：

```
02  W  PIC  99V99.
02  T  PIC  999V9.
02  X  REDEFINES  W  PIC  X (4).
```

是错的，因为 W 和 X 的描述体之间插入了一个 T 的描述体。把第二行与第三行对换就正确了。

（三）可以多次重定义，但必须紧跟出现，而且要求使用最初定义的数据名。如：

```
01  A.
  02  A1  PIC  9 (6).
  02  B1  REDEFINES  A1.
    03  B11  PIC  X (2).
    03  B12  PIC  9 (4).
  02  C1  REDEFINES  A1  PIC  X (6).
  02  D1  REDEFINES  A1.
    D11  PIC        99V99.
    D12  PIC        X (2).
```

这种写法是正确的。B1，C1 和 D1 都对 A1 重定义。如果改成：

```
  ⋮
02  C1  REDEFINES  B1  PIC  X (6).
02  D1  REDEFINES  C1.
  ⋮
```

按标准 COBOL 的规定它是错了（但有些计算机系统 COBOL 允许这样的用法）。

（四）REDEFINES 子句不能用于文件节的 01 层中，因文件节中 01 层描述的是记录。一个多格式记录文件的记录被读入内存后，必然在同一记录区。因此，它的共占内存是必然的、隐含的，不用 REDEFINES 子句来重定义（有关多格式记录文件的概念和使用见本章 §7.6）。

但工作单元节（WORKING-STORAGE SECTION）中的 01 层可以用 REDEFINES 子句重新定义。因这里的 01 层不是指输入输出文件的记录，而是指组合项。

（五）用 REDEFINES 子句可以改变数据的结构，但数据名 1 和数据名 2 的长度应相等。如：

```
02  A  PIC  9 (15).
02  B  REDEFINES  A.
  03  B1  PIC  X (8).
  03  B2  PIC  9 (6).
```

是不对的，因 A 为 15 字节，B 为 14 个字节，重定义不应改变长度。

（六）REDEFINES 子句应在其它子句之前。如：

```
03  A  REDEFINES  T  PIC  X (6)  JUST  RIGHT.
```

*是正确的，而：*

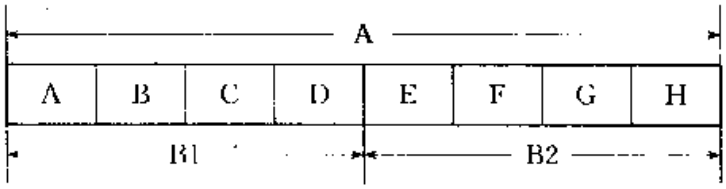
```
03  A  PIC  X (6) JUST  RIGHT  REDEFINES  T.
```

是错误的。

（七）内存中的值为数据名 1 和数据名 2 共享。如：在工作单元节中对 A 赋以初值：

```
02  A  PIC  X (8)  VALUE  'ABCDEFGH'.
02  B    REDEFINES  A.
    03  B1  PIC  X (4).
    03  B2  PIC  A (4).
```

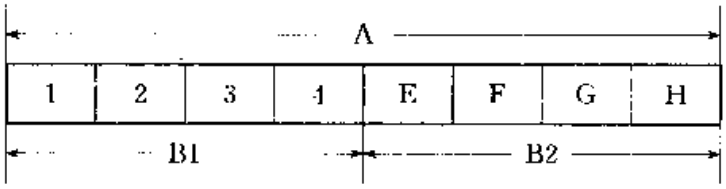
由于 A 中已有值。因此，B1 中也有了确定的值 A B C D，B2 中有确定的值 E F G H，见下图。




在程序执行过程中，用到 A 和 B（包含 B1，B2）都是指向同一部分内存单元。不论 A 或 B 的内容有何改变，都会使 A 和 B 的值同时改变。如执行：

```
MOVE  '1234'  TO  B1
```

则此时内存中情况为：




如果此时将 A 输出，则打印出来的是：1 2 3 4 E F G H。

也就是说，用重定义子句后，数据名 1 和数据名 2 的名称和两种数据结构同时存在，都有效。程序中可用其中任一个。它们在内存中为同一段存储单元。如果改变了内存内容，则二者的值都因而改变（它和 FORTRAN 语言中的等价语句作用类似）。

（八）重定义子句所在的数据描述体中不能使用初值子句赋初值。如：

```
02  T  PIC  9 (4)  VALUE  1234.
02  W  REDEFINES  T  PIC  X (4)  VALUE  'A B C D'.
```

是不对的。因用了重定义后，T 的内容 1 2 3 4 也就是 W 的值，不能再赋一次初值。但可以通过 MOVE 语句或其它执行语句来使 W 和 T 再获新值（而不是初值）。

在第九章（表的建立和查找）中，读者可以看到利用 REDEFINES 语句可以巧妙地给表元素赋初值，解决编程序过程中的某些障碍。

## §7.5 重命名子句（RENAMES 子句）

用 REDEFINES 子句可以在不改变数据项的长度的前提下，重新定义数据区的名称和数据结构的形式（包括重新定义初等项的类型和长度）。用重命名（RENAMES）子句可以把原来已定义的某些数据项重新组合成一个新项，并以一个新名字来代表它。但用重命名子句不能改变原来各初等项的类型、长度等属性。

例如：

```
01  A.
  02  B…
    03  G…
    03  H…
  02  C.
    03  I…
    03  J…
  02  D…
  02  E…
  02  F…
66  K  RENAMES  G  THRU  I.
66  M  RENAMES  B  THRU  C.
66  N  RENAMES  E.
```

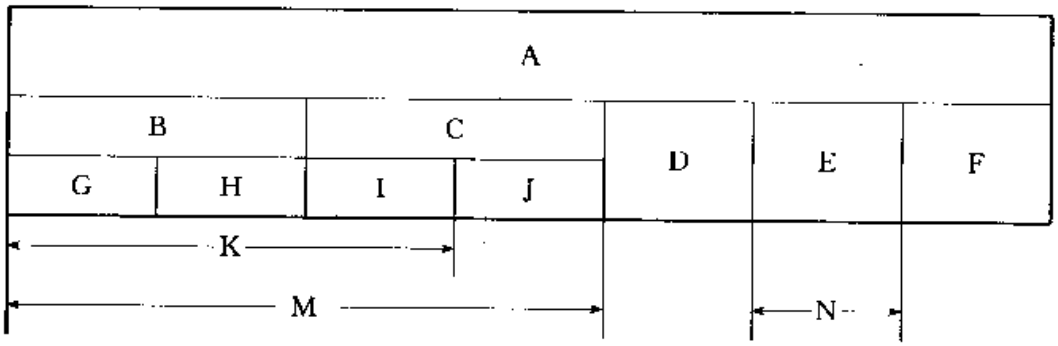



图 7.2

表示将数据项G到I这部分内存区以一个新名K代表。B到C部分以M代表。E部分又名N。K和M是重新组合的组合项，见图7.2。由于它没有改变G，H，I，J原有的数据结构，因此不必在K，M，N描述体中再写PIC子句。只要在层号66之后写上RENAMES子句以说明新名代表的范围即可。此时K为组合项，下属G，H，I三个初等项，M下属两项组合项B，C，它们又下属G，H，I，J四个初等项。

某些初等项按第一种形式的结构组合为一个组合项（G，H组成B），而按另一形式的结构可以形成新的组合项（G，H，I组成K）。两种结构均有效，在程序设计中可以灵活使用RENAMES子句。例如，学生的成绩记录如图7.3所示。

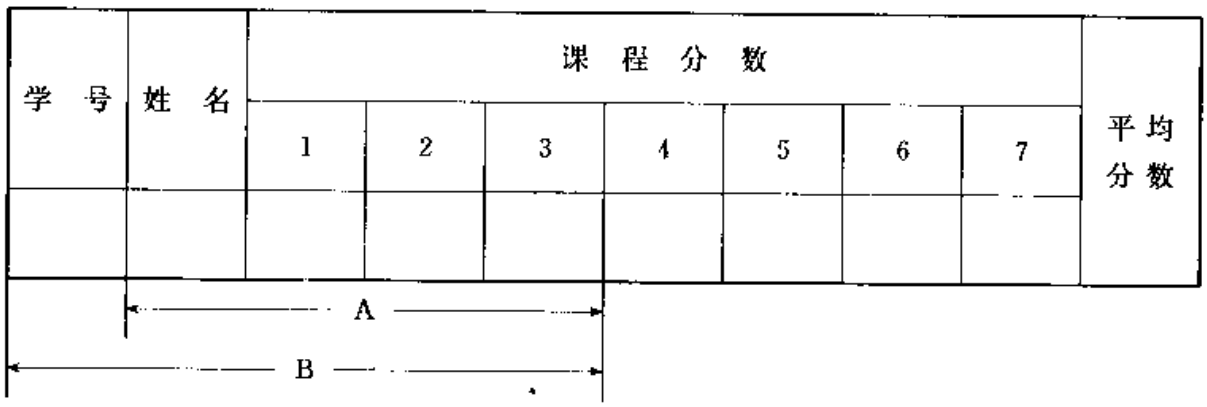



图 7.3

假如已对该记录作了数据描述，但有时并不想按原有结构形式使用。譬如，只想打印出学生姓名和前三门课（数、理、化）成绩。则可用 RENAMES 子句使 A 代表该几项，用 DISPLAY A 则显示出这部分内容。如想打印学号、姓名和数理化成绩，则可用 B 代表这几项，然后 DISPLAY B 即可。这就显得灵活方便。

重命名子句的一般形式：

```
66  数据名1  RENAMES  数据名2  [THRU 数据名3].
```

说明：

（一）层号只能用 66，它必须紧跟在 01 层记录中最后一个数据描述体之后，因为它是对记录中有关部分重新组合和命名的。

数据名 2、数据名 3 可为初等项或组合项，但它们不能是同一个数据名。

（二）如无 THRU 部分，则数据名 1 与数据名 2 代表的是同一内容。如：

```
66  A  RENAMES  B.
```

只重定名，不重新组合。

如用了 THRU 部分，则表示重新进行组合。此时数据名 1 是一组合项，它包括从数据名 2 到数据名 3 的所有初等项。

（三）用 THRU 时，数据名 2 在记录中的位置应在数据名 3 之前，而且数据名 3 不应包括在数据名 2 之中。在本节开始举的例子中，下面的写法是错误的。

```
66  K  RENAMES  I  THRU  G.        (G 在 I 之前)
66  K  RENAMES  C  THRU  J.        (J 是 C 的一部分)
```

（四）RENAMES 子句只能用于工作单元节中，不能用于文件节中。

## §7.6 遇零置空子句（BLANK 子句）

BLANK 子句的作用是：当数据项的值为零时，使它的内容改变为空白。这个子句只能用于数值型或编辑数值型的初等项。

如有：

```
03  A  PIC  $ (5) .99  BLANK  WHEN  ZERO.
03  B  PIC  $Z (3) 99  BLANK  WHEN  ZERO.
03  C  PIC  9999       BLANK  WHEN  ZERO.
```

当执行：MOVE 0 TO A，B，C 后，A，B，C 的值均为 0，由于有 BLANK WHEN ZERO 子句，它们都变为空格（空白）。

但若在同一个数据描述体中用了“*”号来代替高位零，又有 BLANK WHEN ZERO 子句，则 BLANK WHEN ZERO 子句不起作用。如：

```
03  D  PIC  *  (5)  BLANK WHEN ZERO.
```

若执行 MOVE 0 TO D，结果 D 的内容不是空格，而是 * * * * *。

## §7.7 对齐子句（JUSTIFIED 子句）

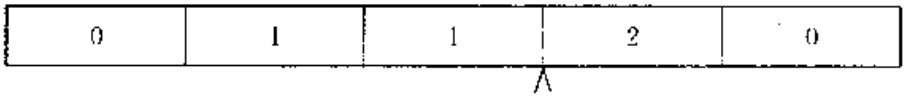
以前已经介绍过，当把一个数据送入一个数据项 C 时，如果 C 是数值型数据项，且其

长度大于送入的数据长度，则按小数点对齐，多余位补以数字零。

如 77 C PIC 9 (3) V 99.

执行：MOVE 11.2 TO C

则 C 的存储情况为：




如果是字符型或字母型数据，则自左面右逐个字符传送，多余位置补空格，接收项长度不够时，将数据右端截断。如：

| 发送项数据 | 接收项描述 | 接收项内容 |
|---|---|---|
| A B C D | X (4) | A B C D |
| A B C D | X (3) | A B C |
| A B C D | X (5) | A B C D ␣ |

这是标准的对齐方式，即字符或字母型数据传送时，采取“左对齐”方式，“向左看齐”。

也可以改变非数值型数据的对齐方式。如果想改为“右对齐”，即“向右看齐”，可以用 JUSTIFIED 子句。先看例子：

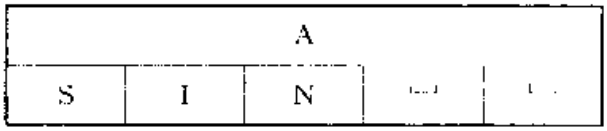
77 A PIC X (5).

77 B PIC X (5) JUSTIFIED RIGHT.

上面 B 描述体中，JUSTIFIED RIGHT 的意思是“右对齐”，如果执行下面一个 MOVE 语句：

MOVE 'SIN' TO A, B

则 A 和 B 的内容分别为

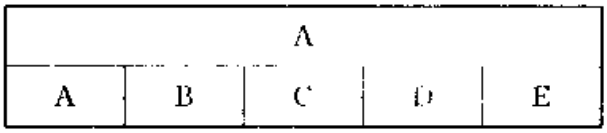


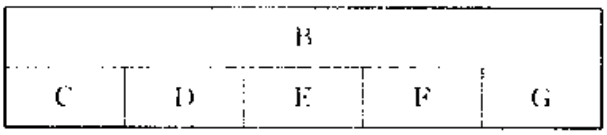

(左对齐)


(右对齐)

如果执行 MOVE'ABCDEFG' TO A, B

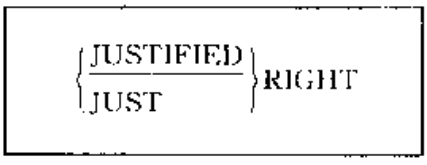
则 A，B 中内容分别为




对 A 的传送是从左边第一个字符开始，一一对应，存入五个字符后多余字符舍去。对 B 是将欲送入的数据的最右边一个字符送到 B 中最右边一个字符，即自右面左，放完五个字符后多余截断舍去。

JUSTIFIED 子句的一般形式为：



JUST 是 JUSTIFIED 的缩写。

说明：JUSTIFIED 子句只能用于字母型或字符型数据项，而不能用于数值型数据项和编辑型数值项，因为后者是按小数点位置对齐的方式定位，而不能用其它方式定位。

## §7.8 同步安置子句（SYNCHRONIZED 子句）

在以前的叙述中，各数据项在内存中是连续地存放的。假若已在工作单元节中定义了以下数据项：

```
01  A.
    02  A1  PIC  9 (3)   VALUE  82.
    02  A2  PIC  X (3)   VALUE  'ABC'.
    02  A3  PIC  9       VALUE  7.
    02  A4  PIC  X (2)   VALUE  'XY'.
    02  A5  PIC  9 (2)   VALUE  12.
```

在内存区中，A1，A2，A3，A4 和 A5 是紧凑相邻的。如图 7.4 示：

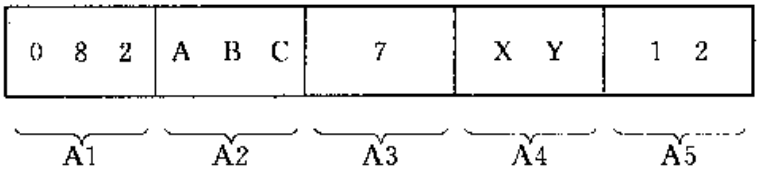



图 7.4

但在许多定字长的计算机中，一个机器字（Word）往往指定为四个字节（也有定为其它长度的）。两个机器字之间为“自然边界”。为了提高效率，从内存取数据时，通常是一次将一个机器字（也可以是半字或双字，由具体计算机系统规定）一起取出来的。因此，按上述存放方法，就会使一个数据项跨两个机器字，即跨越“自然边界”。如图 7.4 表示的数据项 A2 就是一部分在第一个机器字范围内，另一部分在第二个机器字范围内。也可能在一个机器字范围内包含两个甚至多个数据项，如 A2 的一部分、A3 和 A4 的一部分都在第二个机器字内，见图 7.5 所示：

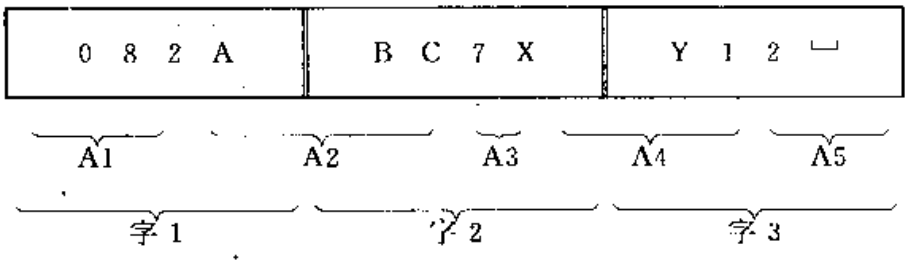



图 7.5

这样虽然排列紧凑，节省了内存空间，但却增加了目标程序的运行时间。因为为了引用某一个数据项，必须先按机器字取出来，并加以重新组织，这就增加了机器的操作。

如果在存放数据时，按“自然边界”安置，即一个机器字内不包含两个或多个数据项，这样就可以在引用数据项时简单迅速，提高操作效率。

用同步安置子句（SYNCHRONIZED 子句）可以指定数据项在内存中如何按自然边界

来安置。

这个子句的一般格式为：

```
{SYNCHRONIZED} [LEFT ]
{SYNC        } [RIGHT]
```

如果用 SYNCHRONIZED LEFT，表示数据项从一个机器字的左边自然边界开始存放，如果这个数据项的长度不是一个机器字，则右边补零（对数值型数据项）或空白（对非数值型数据项），后继的数据项不能使用这些空余的字节，而必须从下一个机器字的左边自然边界开始存放。这些空余的字节称作“填补字节”。如果一个数据项长度大于一个机器字，则连续占用若干个机器字，直到将数据项内容全部放下，最后一个机器字右边自然边界以左的空余字节补零或空格。

如果用 SYNCHRONIZED RIGHT，表示数据项在一个机器字的右边自然边界处终止，即向右靠。左边空余的字节也是用零或空格填补。

如果我们把前面定义的数据项改写为：

```
01  A.
    02  A1  PIC  9 (3)     SYNC     LEFT     VALUE  82.
    02  A2  PIC  X (3)     SYNC     RIGHT    VALUE  'ABC'.
    02  A3  PIC  9         SYNC     LEFT.
    02  A4  PIC  X (2)     VALUE    'XY'.
    02  A5  PIC  9 (2)     VALUE    12.
```

它在内存中的情况如图 7.6 所示：

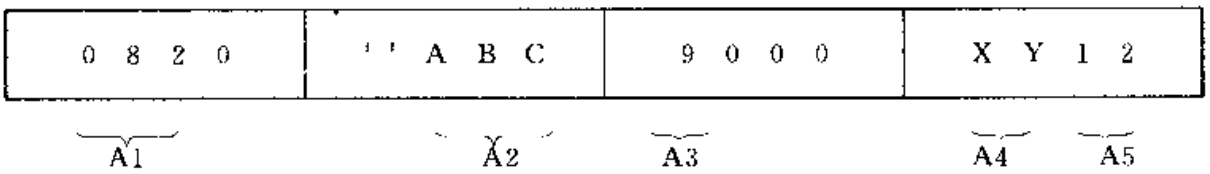



图 7.6

按指定，数据项 A1 从第一个机器字左边界开始存放，但 A1 只有三个字节，故第一个机器字内还有一个多余的字节，用零填补。A2 是向右边界靠，在第二个机器字中左边第一个字节以空格填补。A3 是靠左边界，右边补三个零。A4 和 A5 由于未指定 SYNC 子句，故仍按以前的规定，紧凑相邻地存放在一个机器字中。

当引用带有 SYNC 子句的数据项时，与数据项长度有关的操作（如长度溢出、对齐、截断等）不受 SYNC 子句的影响，仍按 PIC 子句规定的长度处理。譬如将数值 1 2 传送给 A3，A3 中只能放入 2（发生截断），不要认为 A3 已变成 4 个字节而按 PIC 9（4）处理了。

如果一个程序运行时间很长，就应当用 SYNC 子句来减少执行时间，即使要多用一些内存空间也是值得的。在一个反复运行的大程序中，用 USAGE COMP 子句和 SYNC 子句会大大提高运行效率，效果是很显著的。但在初学 COBOL 时，一般是不常用这个子句的。

## §7.9 多格式数据记录——记录区的重叠

每一个文件在内存区都相应地分配一个记录区，该记录区的数据结构应在数据部的文件节中描述。如果从数据文件（如磁盘文件）读入的数据都属于同一记录格式，则每读入一条记录，记录中各个数据都被送入相应的数据项，然后进行运算或其它处理。但有时从文件中读入的数据不是属于同一种格式的，它们代表不同的内容。例如：在人员情况的数据文件中，包含两类记录。一些记录中包含如下的地址数据：

| (人员号) NUMB | (标记) FLAG | (姓名) NAME | (路名和号码) ROAD | (城市) CITY |
|---|---|---|---|---|

而另一些记录中包含有职工工作时间的统计：

| (人员号) NUM | (标记) FAG | (日期) DATE | (星期一) MON | (星期二) TUE | (星期三) WED | (星期四) THU | (星期五) FRI | (星期六) SAT | (星期日) SUN |
|---|---|---|---|---|---|---|---|---|---|

假如这些记录同属一个数据文件，在送入内存后它们是放入同一个文件记录区中的，那么，怎样区别送入的数据是哪一类数据并作相应的处理呢？

我们先在数据部的文件节中描述它们：

```
FD  IN FILE  LABLE  RECORD  IS  STANDARD.
    DATA  RECORDS  ARE  ADDRESS-CARD. HOURS CARD.
01  ADDRESS-DATA.
  02  NUMB  PIC  9 (5).
  02  FLAG  PIC  9.
  02  NAME  PIC  X (20).
  02  ROAD  PIC  X (20).
  02  CITY  PIC  X (20).
01  HOURS-DATA.
  02  NUM       PIC  X (5).
  02  FAG       PIC  9.
  02  DATE      PIC  9 (6).
  02  MON       PIC  99V9.
  02  THU       PIC  99V9.
  02  WED       PIC  99V9.
  02  THU       PIC  99V9.
  02  FRI       PIC  99V9.
  02  SAT       PIC  99V9.
  02  SUN       PIC  99V9.
```

文件 IN-FILE 包含两种格式的记录：ADDRESS-DATA 和 HOURS-DATA。但读入记录时，一律送入同一记录区。为了区分送入的数据是哪一种格式的记录，在写程序时可以人为地设一个标志。本例中我们可以假定 ADDRESS-DATA 记录中的数据项 FLAG 等于 1，而 HOURS-DATA 记录中的数据项 FAG 等于 2。（"1" 和 "2" 是用户在设计程序时自己

选定的标志，也可以选择其它数字)。然后在读入记录后，用下面的语句判断读入的是哪一类记录。

IF FLAG 1 ……

如果 FLAG＝1，则表明是 ADDRESS-DATA 记录，如不等于 1（而等于 2），则为 HOURS-DATA 记录。根据此可作不同的处理。

那么，用 FAG 作判断是否可以？也可以。如用：

IF FAG 1……

作用也一样。

在数据部中定义了文件 IN-FILE 有两种格式的记录，用两个记录描述体（有两个 01 层）来描述其数据结构。它的作用与§7.4 中重定义子句（REDEFINES 子句）相似。即在同一内存区中，它有两种数据结构和两套数据名，二者都有效。因此第 6 个字节位置的数据名既叫 FLAG，又叫 FAG。无论是 IF FLAG＝1 或 IF FAG＝1 都能对此字节的实际内容作出判断。

如果两个不同格式的记录长度不同，则内存记录区的长度按长者确定。

也可以不必设两个不同的数据名作标志（如 FLAG 和 FAG），只需设一个 FLAG，而在第二类记录中的相应位置上以 FILLER 来代替。读入记录后，只要用 IF FLAG＝1…来判断，比较方便，见图 7.7。


图 7.7

能否在两类记录中都用同一个数据名 FLAG 作判别用？我们说，不好。因为如果两处都用 FLAG，则 FLAG 就不是唯一的名字了，程序中如写 IF FLAG＝1…就不行。需要具体指出是哪一个 FLAG 才行，要加上限定词，如 IF FLAG OF ADDRESS-DATA＝1…或 IF FLAG OF HOURS-DATA＝2…才行，反而不便。因此，只要设一个名字即可，另一处用 FILLER 代替，比较省事。

下面举一个程序例子，说明多格式记录的应用。

**【例 7.1】**

```
IDENTIFICATION  DIVISION.                          (标识部)
PROGRAM-ID.     EX7.1.
ENVIRONMENT     DIVISION.                          (环境部)
INPUT OUTPUT    SECTION.
FILE-CONTROL.
    SELECT  FILE-X  ASSIGN  TO  U01.
    SELECT  FILE Y  ASSIGN  TO  U02.
DATA      DIVISION.                                (数据部)
FILE  SECTION.
```

```
FD  FILE-X  LABLE  RECORD  IS  STANDARD
    DATA RECORD ARE INPUT-A, INPUT-B.
01  INPUT-A.
  02  RECORD-CODE PIC  X.                    (记录代码)
  02  FILLER  PIC XX.
  02  PART-NUMBER PIC  9 (5).                (零件号)
  02  FILLER  PIC XX.
  02  QUANTITY  PIC  9 (6).                  (数量)
  02  FILLER  PIC XX.
  02  PRICE  PIC  99V99.                     (单价)
01  INPUT-B.
  02  FILLER  PIC  X.                        (记录代码)
  02  FILLER  PIC  XX.
  02  CUSTOMER-NAME  PIC  X (20).            (买主名)
  02  FILLER  PIC  XX.
  02  CUSTOMER-NUMBER  PIC  9 (6).           (买主号)
  02  FILLER  PIC  XX.
  02  PART-NUM  PIC  9 (6).                  (零件号)
  02  FILLER  PIC  XX.
  02  QUANTITY-OF-SALES  PIC  9 (4).         (销售数)
FD  FILE-Y  LABLE  RECORD  IS  STANDARD
      DATA  RECORD  IS  RECORD-Y.
01  RECORD-Y.
  02  FILLER  PIC  X.
  02  PRINT LINE  PIC  X (136).
PROCEDURE  DIVISION.                          (过程部)
A.      OPEN INPUT FILE X
          OUTPUT FILE-Y.
B.      READ FILE-X AT END GO TO C.
    IF  RECORD-CODE = 'B'
      MOVE  INPUT B  TO  PRINT-LINE
      WRITE  RECORD-Y  AFTER  2.
    GO  TO  B.
C.  CLOSE FILE-X FILE-Y.
    STOP RUN.
```

此程序中有两类记录，一类是产品记录（包括零件号、数量、单价），一类是买主的记录（包括买主名、买主号、零件号、销售数）。每个记录中的第一个数据项设标志为“A”或“B”（“A”表示为产品记录，“B”为买主记录）。要求将记录读入后，找出买主的记录并把它打印出来，如果读入的是产品记录不打印出来。

## §7.10 复写语句（COPY 语句）

COBOL 程序中的 DATA DIVISION 部分往往是很长的，因为其中包含许多的数据项

的描述。但是，不同文件所用到的数据描述往往有些是相同的或类似的。譬如主程序和子程序往往可以共用同一个数据描述。有些程序中的数据描述和另一程序的差不多，只需作些小的修改或补充。这时，就可以不必要求每一个程序设计人员重复地进行这部分工作，甚至连照抄都没有必要。利用 COPY 语句可以使某些记录描述和数据描述为不同的程序共用。为此要建立一个“源程序库”，将上述这些共同使用的源程序中的某一部分事先存入“库”中。程序设计人员在写源程序时用 COPY 语句就可以将这些源程序体插入到自己的源程序中。

例如，假如我们事先已编好一个“源程序体”（源程序中的一部分）放入库中，对这个放入库中的源程序体（称为“库文本”）起一个名字为“LIB1”。它的内容如下：

```
01  A.
  02  A1  PIC  9 (4).
  02  A2  PIC  9 (6) V99.
  02  A3  OCCURS  5  TIMES  PIC  X (10).
  02  A4  PIC  X (8).
```

在我们的程序中如果想用以上形式的描述，不必一一照写，只要写成下面形式即可：

```
01  B  COPY  LIB1.
```

在编译这个 COPY 语句时，将“LIB1”库文本复制插入到程序中，按以下结果进行编译。

```
01  B.
  02  A1  PIC  9 (4).
  02  A2  PIC  9 (6) V99.
  02  A3  PIC  OCCURS  5  TIMES  PIC  X (10).
  02  A4  PIC  X (8).
```

注意，层号 01 后的记录名 B 并不改变为 A。因为它在 COPY 作用范围之外。

还可以对复制过来的内容作某些修改，如想将 A1～A4 改变为 B1～B4，可以用 REPLACING 子句作“取代”的处理。如：

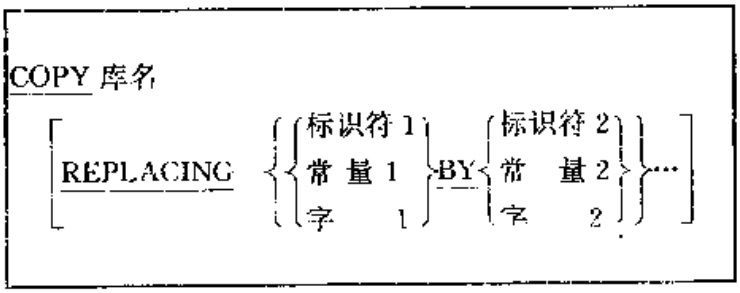
```
01  B  COPY  LIB1
       REPLACING  A1  BY  B1
                  A2  BY  B2
                  A3  BY  B3
                  A4  BY  B4.
```

则在复制时将库文本中的“A1”改为“B1”，“A2”改为“B2”……。在编译此 COPY 语句时，和编译以下的内容一样：

```
01  B.
  02  B1  PIC  9 (4).
  02  B2  PIC  9 (6) V99.
  02  B3  PIC  OCCURS  5  TIMES  PIC  X (10).
  02  B4  PIC  X (8).
```

但库文本内容不因此被改变。

COPY 语句的一般格式为：




在此“一般格式”中，“字”的含义为：不超过 30 个字符组成的字符序列。它包括数据名，条件名，过程名，表意常量，助忆名以及保留字等。

在有的系统中，一个库中不止放一个“源程序体”，而是放若干个“源程序体”，每一个“源程序体”称为这个库的一个“成员”。譬如在库名为 A 的库中，有一个名为 A1 的源程序体（即库中成员），如果要复制它到 COBOL 源程序中，可以写成以下形式：

01 B COPY A1 OF A.

“A1 OF A”表示“A 库中的名为 A1 的成员”，它代表一个“源程序体”。

在数据部中 COPY 语句必须按以下形式书写：

(1) 在文件节中

FD 文件名 COPY 语句.

SD 排序文件名 COPY 语句.

01 数据名 COPY 语句.

(2) 在工作单元节和联接节中

77 数据名 COPY 语句.

01 数据名 COPY 语句.

77 数据名 1 REDEFINES 数据名 2 COPY 语句.

01 数据名 1 REDEFINES 数据名 2 COPY 语句.

(3) 在报表节中

RD 报表名 COPY 语句.

RD 报表名 WITH CODE { 常 量 / 助忆名 } COPY 语句.

01 数据名 COPY 语句.

事实上，COPY 语句不仅可用于数据部，而且可用于其它部分。如：环境部中：

SOURCE-COMPUTER. COPY 语句.

OBJECT-COMPUTER. COPY 语句.

SPECIAL-NAMES. COPY 语句.

FILE-CONTROL. COPY 语句.

I-O-CONTROL. COPY 语句.

SELECT 文件名 COPY 语句.

但在环境部用 COPY 语句的优越性并不显著。

在过程中也可以用 COPY 语句，如：

节名 SECTION. COPY 语句.

段名. COPY 语句.

使用源程序库和 COPY 语句，有利于在不同文件中用标准化的记录书写程序，减少编写程序的工作量，便于程序的保存和修改。

# 习　题

7.1　下面的写法对不对？

(1) 77 A PIC X (20) USAGE DISPLAY.

(2) 77 B PIC X (15) USAGE COMP.

(3) 01 C PIC 9 (4) COMP-2. (COMP-2 指内部长浮点形式)

(4) 01 D COMP.
　　02 D1 PIC 9 (6).
　　02 D2 PIC 9 (4).

(5) 02 F COMP.
　　04 F1 PIC 9 (6) COMP-2.
　　04 F2 PIC 9 (5).

7.2　如 A，B，C，D 已作以下描述：

77 A PIC X (4).
77 B PIC X (4) JUSTIFIED RIGHT.
77 C PIC X (6) JUSTIFIED RIGHT.

当执行：

MOVE 'CHINA' TO A, B, C

后，A，B，C，D 中的内容是什么？

7.3　已有：

02 ABC.
　　03 A PIC 9 (6).
　　03 B PIC X (4).
　　03 C PIC A (5).
02 DEF REDEFINES ABC.
　　04 D PIC X (2).
　　04 E PIC A (6).
　　04 F.
　　05 F1 PIC 9 (2).
　　05 F2 PIC X (5).

用图表示内存中数据存放的情况。

7.4　已有：

01 A.
　　02 A1 PIC X (4).
　　02 A2.
　　03 A21 PIC X (2).
　　03 A22 PIC 9 (4).
　　02 A3 PIC X (6).
　　02 A4.
　　04 A41 PIC X (3).

```
    04  A42  PIC  9 (6).
66  X  RENAMES  A2.
66  Y  RENAMES  A21  THRU  A3.
66  Z  RENAMES  A22  THRU  A42.
66  T  RENAMES  A1  THRU  A22.
```

用图表示内存中数据存放的情况。

7.5 什么叫多格式记录？在读入一个记录后如何区别它们？如果一个文件中只有一种格式的记录，要不要设区别标志？

7.6 不同格式的记录是否要求相同的长度？如果长度不同，按哪个长度分配内存记录区？

# 第八章　子程序

## §8.1　概　　述

用已介绍过的执行语句（PERFORM 语句）可以使某一语句序列（例如一个或多个节或段）重复执行若干次。但这种调用方式只限于在本程序中使用。每一个程序都有自己的名字和四大部分。PERFORM 语句所调用的语句序列必须也在以该名字命名的程序中，也就是说，如果有一个程序名为 A，另一程序名为 B，则在 A 程序中的 PERFORM 语句只能调用程序 A 中的过程部的语句序列，而不能调用 B 程序中的语句。

如果要调用的部分比较复杂，则会使程序冗长、庞杂，使编制者和阅读者都会感到不便，我们可以把这部分语句序列不放在本程序中，而另外编成一个程序，就是子程序，需要时调用这个子程序。我们把原来的程序称主程序。也就是说，由主程序根据需要调用子程序。特别当被调用的部分不仅是一个程序需要调用它，而且许多个程序都需调用它时，把它单独编成子程序更为方便，可以为各程序共用。

为了说明子程序的概念，先举一简单的例子。

**【例 8.1】**　编一个打印一行“*”符号的子程序。

先编一主程序，主程序每调用一次子程序，就打印出一行星号。

（一）主程序：（只写与调用子程序有关的部分）

```
IDENTIFICATION DIVISION.        (标识部)
PROGRAM-ID. A.                  (程序名为 A)
ENVIRONMENT DIVISION.           (环境部)
  ⋮
DATA DIVISION.                  (数据部)
  ⋮
PROCEDURE DIVISION.             (过程部)
  ⋮
   CALL    B.                   (调用子程序 B)
  ⋮
```

（二）子程序

```
IDENTIFICATION DIVISION.        (标识部)
PROGRAM-ID. B.                  (程序名为 B)
ENVIRONMENT DIVISION.           (环境部)
DATA DIVISION.                  (数据部)
WORKING-STORAGE SECTION.
77    X    PIC  X (80).
PROCEDURE DIVISION.             (过程部)
MOVE  ALL  '*'  TO  X.
DISPLAY  X.
```

EXIT PROGRAM.　　(子程序出口)

执行A程序。当执行到CALL B语句时，就调用B程序（子程序）。CALL语句称为"调用子程序语句"，CALL B表示调用程序名为B的子程序。此时，不再往下执行主程序的语句，而是找到B程序后，转而执行B程序，直到遇到B程序中的EXIT PROGRAM语句（子程序出口语句），表示子程序执行完毕，返回到主程序，然后继续执行主程序中CALL语句下面的过程部语句。见图8.1。

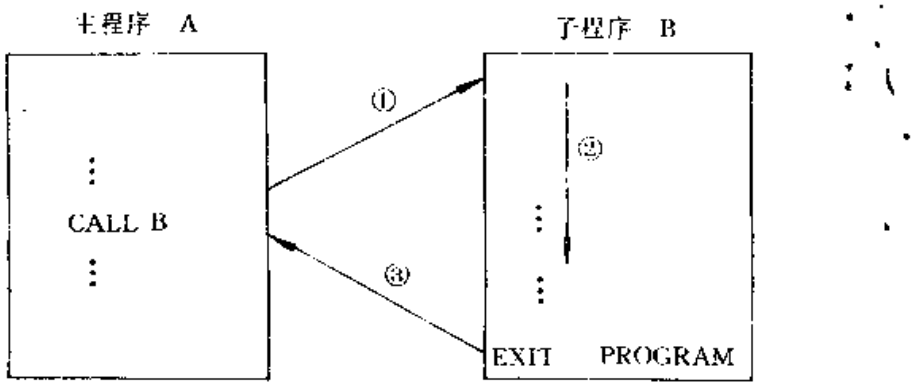



图 8.1

可以看到程序A和程序B分别是两个程序，各有自己的程序名，都有四大部分。子程序B的结构和一般程序基本相同（只有某些部分不同，这将在本章§8.3中介绍），它可被任何一个程序调用。

通过此例，可以看到在程序的标识部中给程序命名的作用。调用子程序时就是按此名字查找的。

主程序可以调用子程序，而子程序不能调用主程序。子程序还可以再调用其它子程序，即子程序可以嵌套，见图8.2。

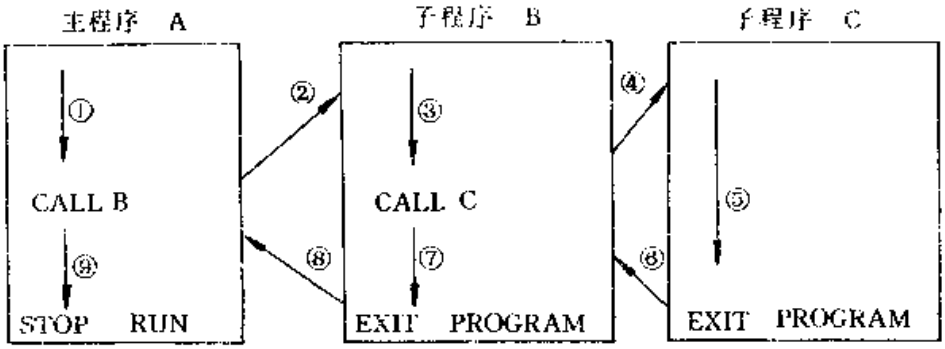



图 8.2

其执行过程为：①先执行主程序中CALL B以前的语句。②遇CALL B语句，转到B程序。③执行B程序中过程部中CALL语句以前的语句。④遇CALL C，转到子程序C。⑤执行子程序C。⑥遇子程序C中的EXIT PROGRAM语句，返回调用子程序C的程序（即B程序）中CALL C下一语句处。⑦执行子程序B中CALL C下面的各语句。⑧遇子程序B中的EXIT PROGRAM语句，返回调用子程序B的程序（即主程序A）中CALL B语句的下一语句。⑨继续执行主程序A中CALL B后面的语句，直到结束。

注意：不应在子程序C中写CALL C或CALL B或CALL A。因为这是自己调用自己或“被调用程序”调用一个“调用程序”(这种直接或间接调用自己的方式，称为递归调用。COBOL语言不允许递归调用)。

在程序设计中使用子程序的优点：

(1) 可以不必写一个庞大的主程序，而根据需要建立一个主程序和一批子程序。每一子程序完成一定的功能，由主程序分别调用子程序。这样可使主程序简化，程序的结构层次清晰。

(2) 由于每一个子程序只完成一个功能，任务单纯，因此易于编写，易于调试，可减少错误。

(3) 由于子程序有相对的独立性，可以将一个大的程序分为主程序和若干个子程序，分别由若干人完成。便于多个程序员合作共同编制一个复杂的程序。

(4) 修改子程序比较容易。如果某一个子程序中需要修改一些处，只需修改此子程序即可，而不必重新编译所有的程序，节省编译时间。

(5) 通用性强。将常用的一些程序写成子程序，供各用户使用，可减少重复劳动。

使用子程序是程序设计的一种很有用的方法。结构化程序设计的一个重要内容是“模块化设计”，一个软件可划分为若干个功能模块，在程序中用子程序实现一个模块的功能。

## §8.2 调用程序与被调用程序间的数据联系

调用程序(可以是主程序或调用其它子程序的子程序)和被调用程序间的数据之间往往需要发生某些联系。这就是程序间的数据联系。

在例8.1中，没有数据的传递，子程序的作用只是打印出一行星号，子程序中的数据不传回主程序。但在有的程序中需要使不同程序中的数据之间建立联系，即存在数据的传递。由主程序将数据传递到子程序，或从子程序传递回主程序。我们通过下面简单的程序片断来说明这一点。

**【例8.2】**

主程序：

```
    ⋮
MOVE    2    TO    A.
MOVE    8    TO    B.
CALL SUB USING  A, B, C.
MOVE C TO T.
DISPLAY C.
    ⋮
```

子程序SUB：

```
    ⋮
PROCEDURE DIVISION USING  X, Y, Z.
COMPUTE  Z = X + Y.
    ⋮
```

EXIT PROGRAM.

主程序中 CALL 语句中用了 USING 子句，它包括参数 A，B，C。A 和 B 的值是在主程序中给出的，C 的值未给出。在执行 CALL 语句时，将主程序的 A，B 的值（2 和 8）传送给子程序的 X 和 Y。执行子程序中的计算（COMPUTE）语句，计算出 Z 的值，这个 Z 的值，在执行完子程序返回主程序时，传递给主程序中的 C。现在在主程序中 A 的值为 2，B 值为 8，执行了程序时 X 就等于 2，Y 就等于 8，子程序计算出 Z 等于 10。返回主程序后执行 DISPLAY C，将显示出 C 的值 10。

调用语句的一般格式为：

```
CALL  子程序名  [USING  数据名 1  [，数据名 2] …]
```

被调用程序中过程部部头的一般格式为：

```
PROCEDURE  DIVISION  [USING  数据名 1 [，数据名 2] …]
```

说明：

(1) 主程序中除了 CALL 语句外，其它与以前介绍过的没有什么不同。

(2) 主程序中 CALL 语句中 USING 子句中用到的参数（数据名 1，数据名 2，…）的个数和子程序中过程部部头 USING 子句用到的参数个数必须相等。如上例中主程序 CALL 语句中用到三个参数 A，B，C，子程序过程部部头中 USING 子句后 X、Y、Z 也是三个参数。如不相等，则出错。

(3) 主程序和子程序中 USING 子句中各参数是依照它们各自的次序确定关系的，而不是依照名字相同来确定对应关系的。如：

主程序：

```
CALL    T  USING  A，B，C，D.
  ⋮
```

子程序：

```
PROCEDURE DIVISION USING  X，  C，  A，  Y.
  ⋮
```

它们的对应关系是：

```
主程序                子程序
  A←- - - -
```

程序期间主程序和子程序中相应的数据名（可以是不同名，也可以同名）共享内存中同一段存储单元，因此它们具有同一个值（见下表）。

| A，X | B，Y | C，Y |
|---|---|---|
| 2 | 8 | 10 |

当执行完子程序后，主程序与子程序之间的这种关系就不存在了。例如，在主程序中若在 CALL 语句之后执行“MOVE 20 TO A”，A 的值 20 不会传递到子程序的 X 中去。

（5）调用程序和被调用程序的相应的数据项的长度应相同，如主程序中对 A 的描述用 PIC 99，则子程序中对 X 的描述可以用 PIC 99，或 PIC XX 等。如果长度一样而描述的类型不同，则按以前介绍过的传送的规则进行类型转换。

（6）也可以无数据的传递（如例 8.1）。在调用子程序时，子程序按其本身的语句内容执行，而不将数据带回主程序。如：

**【例 8.3】**

主程序：

⋮

```
CALL A.
```

⋮

子程序 A：

⋮

```
PROCEDURE   DIVISION.
   MOVE   1   TO   A.
   MOVE   2   TO   B.
   ADD    A   TO   B.
   DISPLAY   B.
   EXIT PROGRAM.
```

子程序中数据项 A，B 的值不由主程序传送，而由子程序本身提供，计算结果（在 B 中）也不送回主程序，在子程序中将 B 值显示出来。

（7）每个程序（主程序或子程序）中定义的数据名，只在本程序中有效，例如在例8.2的主程序中不能用 DISPLAY Z 来直接显示子程序中的数据项 Z 的值。只能将 Z 的值传递给主程序中的 C，然后 DISPLAY C。

## §8.3 子程序的结构

子程序同样由四大部分组成：

（一）标识部

在标识部中说明子程序的名字，以供调用程序按名字调用它。此程序名应该是唯一的，即不能与其它程序重名。

（二）环境部

指出子程序运行的环境。子程序中如果用到输入输出文件，那么应在子程序的环境部的

输入输出节中给该文件分配设备，即使子程序用的设备与主程序相同（如都用打印机），也应在子程序中单独说明。

（三）数据部

在子程序中用到的数据项有两种情况：一是与调用程序无关的数据项，一是与调用程序有联系的数据项。例8.2子程序 SUB 中的 X，Y，Z 就是与主程序发生数据联系的数据项。

在子程序中专门设一个联接节(LINKAGE SECTION)，用来说明与调用程序有数据联系的数据项。下面将例8.2加以补充，写成一个完整的程序。

**【例 8.4】**

主程序：

```
IDENTIFICATION    DIVISION.                    (标识部)
PROGRAM-ID. EX84                               (程序名为 'EX84')
ENVIRONMENT       DIVISION.                    (环境部)
DATA              DIVISION.                    (数据部)
WORKING-STORAGE   SECTION.
77  A   PIC  99  VALUE  2.
77  B   PIC  99  VALUE  3.
77  C   PIC  99.
PROCEDURE         DIVISION.                    (过程部)
S.    CALL  'SUB'  USING  A, B, C.             (调用子程序 'SUB' 的语句)
      DISPLAY'C=',  C.
      STOP RUN.
```

子程序：

```
IDENTIFICATION    DIVISION.                    (标识部)
PROGRAM-ID.   SUB.                             (程序名为 'SUB')
ENVIRONMENT       DIVISION.                    (环境部)
DATA              DIVISION.                    (数据部)
WORKING-STORAGE SECTION.                       (工作单元节)
77  N   PIC  99  VALUE  5.
LINKAGE   SECTION.                             (联接节)
77  X   PIC  99.
77  Y   PIC  99.
77  Z   PIC  99.
PROCEDURE         DIVISION USING X, Y, Z.      (过程部)
S.    COMPUTE Z=X+Y.
      DIVIDE N INTO Z.
      EXIT PROGRAM.
```

在主程序中使 A=2，B=3，通过 CALL 语句中的 USING A，B，C 与子程序的 X，Y，Z 建立相应的数据传递的关系。COMPUTE 语句算出 Z=2+3=5。然后除以 N，N 的值为5，所以得 Z=1。子程序中所用到的 N，与主程序无关，它的值是子程序本身提供的。这个 N 应在子程序的数据部中的工作单元节中说明。而 X，Y，Z 是与主程序（调用程序）有联系的数据项，因此应在联接节中说明。

联接节中的数据项描述体同样分别用77层号或01～49层号开头。如以下形式：

```
LINKAGE      SECTION.
77  X  PIC  99.
01  A.
  02  A1  PIC  99.
  02  A2  PIC  99.
```

至今为止，我们已介绍过在数据部中有三个节：

(1) 文件节（FILE　SECTION）用来描述输入输出文件中的数据项。

(2) 工作单元节（WORKING—STORAGE　SECTION）用来描述非输入输出的数据项，譬如中间数据项。

(3) 联接节（LINKAGE　SECTION）用来描述被调用程序（子程序）中与调用程序有联系的数据项。

如果子程序过程部部头的 USING 子句中有数据名，则此数据名应在本子程序中的 LINKAGE SECTION 中加以说明。联接节只能出现在被调用的程序中，但被调用程序并非必须有联接节。对联接节中的数据不另分配存储空间，在调用被调用程序时，联接节中的数据项使用了调用程序相应数据项的存储区。

如果在联接节中描述的数据项多于 USING 子句中数据名的个数，则在子程序中实际能用的以过程部部头的 USING 子句中指出的数据名为限。

(四) 过程部

过程部的部头：

PROCEDURE DIVISION USING 数据名1，数据名2，…。

过程部中应包括一个程序出口语句：

```
EXIT PROGRAM.
```

而不应写成 STOP RUN，否则不能从子程序返回调用程序。

EXIT PROGRAM 可以是过程部的最后一行，也可以不在最后一行，但应该是逻辑上最后一个语句，即在它后面的语句都不被执行。

## §8.4 程 序 举 例

**【例 8.5】** 计算体积，在主程序中给出长方体的长 A，宽 B，高 H。要求由子程序计算出体积并显示出来（从主程序输入多组数据）。

主程序：

```
IDENTIFICATION      DIVISION.
PROGRAM-ID.  EX85.
ENVIRONMENT         DIVISION.
DATA          DIVISION.
WORKING-STORAGE  SECTION.
77  A  PIC  99V99.
77  B  PIC  99V99.
77  H  PIC  99V99.
```

```
77  V  PIC  9 (6) .99.
PROCEDURE              DIVISION.
S.    DISPLAY  'A, B, H=?'.
      ACCEPT  A.
      ACCEPT  B.
      ACCEPT  H.
      IF H=0 DISPLAY 'END' STOP RUN.
      CALL  'SUB' USING A, B, H, V.
      DISPLAY 'V=', V.
      GO TO S.
```

子程序：

```
IDENTIFICATION        DIVISION.
PROGRAM-ID. SUB.
ENVIRONMENT           DIVISION.
DATA        DIVISION.
LINKAGE  SECTION.
77  A  PIC  99V99.
77  B  PIC  99V99.
77  H  PIC  99V99.
77  V  PIC  9 (6) .99.
PROCEDURE              DIVISION  USING  A, B, H, V.
S.    COMPUTE  V=A*B*H.
      EXIT PROGRAM.
```

运行情况如下：

```
A, B, H  ?
0500↙
0500↙
0500↙
V=000125.00
A, B, H=?
0200↙
0200↙
0200↙
V=000008.00
A, B, H=?
0300↙
0300↙
0000↙
END
```

说明：

(1) 程序开始执行后，由主程序显示出“A，B，H=?”，以提示程序员输入A，B和H的值。如果系统确定ACCEPT语句隐含的输入设备是键盘，则通过键盘输入A，B，H的

值。每输入一组数据，调用一次子程序 SUB，计算出体积 V。在主程序中用 DISPLAY 语句显示出 V 的值。

(2) 每执行一次主程序中的 DISPLAY 语句后，转回到 S 段重新执行一次，再输入一组数据，并重复以上过程，直到输入数据中的 H 的值为零为止，此时表示不再需要计算了(这是在设计程序时先考虑好的“停止运算”的条件)，显示出“END”后停止运行。

(3) 我们在主程序和子程序中用了两组相同的数据名 A，B，H 和 V（也可以用不同的名字)。但应注意，主程序中的 A 和子程序中的 A 尽管同名，但如不用 USING 子句使它们建立对应关系，它们之间是没有什么关系的。并不因它们同名而自动传递数据。不同程序中的数据项即使同名也不表示它们代表的是同一个对象。在本例的子程序中，也可以不用 A，B，H，V，而用 A1，B1，H1，V1或其它数据名，效果完全相同。

(4) 在本例中，主程序中的 A，B，H，V 和子程序中的 A，B，H，V 的描述是完全相同的，以保证数据传递正确。

**【例 8.6】** 输入一个班每个学生的姓名、学号和五门课的成绩，要求将学生姓名和每个学生的平均成绩输出到一个文件中保存，将全班各门课的总平均成绩在显示器显示出来。

主程序：

```
IDENTIFICATION      DIVISION.          (标识部)
PROGRAM-ID. EX86.
ENVIRONMENT         DIVISION.          (环境部)
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  IN-FILE    ASSIGN  TO  STDT.
    SELECT  PRINT-FILE  ASSIGN  TO  P-FILE.
DATA                   DIVISION.       (数据部)
FILE  SECTION.
FD  IN-FILE LABEL RECORD IS STANDARD. (输入文件名为 IN-FILE)
01    STUDENT-RECORD.                  (记录名为 STUDENT-RECORD)
  02  NAME  PIC  X (10).              (名字)
  02  NUMB  PIC  9 (6).               (学号)
  02  GRADE-1  PIC  9 (3).            (成绩1)
  02  GRADE-2  PIC  9 (3).            (成绩2)
  02  GRADE-3  PIC  9 (3).            (成绩3)
  02  GRADE-4  PIC  9 (3).            (成绩4)
  02  GRADE-5  PIC  9 (3).            (成绩5)
FD  PRINT-FILE  LABEL  RECORD  IS  STANDARD. (输出文件名 PRINT-FILE)
01  PRINT-LINE.                       (记录名 PRINT-LINE)
    02  FILLER  PIC  X.
    02  NAME-P  PIC  X (10).          (姓名)
    02  FILLER  PIC  X (5).
    02  AVER-P  PIC  9 (3).           (个人平均成绩)
WORKING-STORAGE  SECTION.
77  N  PIC  99  VALUE  0.             (人数)
```

```
77  TOTAL  PIC  9 (6)   VALUE  0.        (班总分)
77  TOTAL-M  PIC  9 (3) V99.            (个人总分)
77  AVERAGE  PIC  9 (3). 99  BLANK WHEN ZERO. (总平均)
77  AVER-M  PIC  9 (3).                 (个人平均成绩)
PROCEDURE        DIVISION.              (过程部)
S.   OPEN  INPUT  IN-FILE
           OUTPUT  PRINT-FILE.
R.   READ  IN-FILE  AT  END  GO  TO  C.
     CALL  'CALCUL'  USING  STUDENT-RECORD, TOTAL-M, AVER-M.
     MOVE  NAME  TO  NAME-P.
     MOVE  AVER-M  TO  AVER-P.
     WRITE PRINT-LINE AFTER 2.           (输出姓名和9人平均成绩)
     ADD  1  TO  N.
     ADD  TOTAL-M  TO  TOTAL.            (累加总成绩)
     GO  TO  R.
C.   COMPUTE  AVERAGE ROUNDED=TOTAL/ (5 * N). (计算总平均成绩)
     DISPLAY  'AVERAGE=',  AVERAGE.
E.   CLOSE  IN-FILE, PRINT-FILE.
     STOP  RUN.
```

子程序：

```
IDENTIFICATION       DIVISION.          (标识部)
PROGRAM-ID. CALCUL.                     (子程序名 'CALCUL')
ENVIRONMENT          DIVISION.          (环境部)
DATA       DIVISION.                    (数据部)
LINKAGE  SECTION.                       (联接节)
77  TOTAL-S  PIC  9 (3).
77  AVER-S   PIC  9 (3).
01  STUDENT-RECORD.
    02  NAME-S  PIC  X (10).
    02  NUMB-S  PIC  9 (6).
    02  GRADE-1-S  PIC  9 (3).
    02  GRADE-2-S  PIC  9 (3).
    02  GRADE-3-S  PIC  9 (3).
    02  GRADE-4-S  PIC  9 (3).
    02  GRADE-5-S  PIC  9 (3).
PROCEDURE    DIVISION USING STUDENT-RECORD, TOTAL-S, AVER-S.
S.   COMPUTE  TOTAL-S=GRADE-1-S+GRADE-2-S+GRADE-3-S
                 +GRADE-4-S+GRADE-5-S.        (计算个人总分)
     DIVIDE  5  INTO  TOTAL-S  GIVING  AVER-S. (计算个人平均成绩)
     EXIT PROGRAM.
```

输入文件（学生成绩数据文件）STDT. DAT

```
A A A A A A A A A A0000010990870340560 99
B B B B B B B B B B0000020450670890670 87
```

C C C C C C C C C C0000031000780890890 89
D D D D D D D D D D0000040780890780670 67
E E E E E E E E E E0000050890890780890 78
F F F F F F F F F F0000060890780670890 67
G G G G G G G G G G0000070890780670780 68
H H H H H H H H H H0000080890780670780 89
I I I I I I I I I I 0000091001000890780 89
J J J J J J J J J J 0000100980870870980 97

输出数据文件 P-FILE. DAT（每个学生的各门课的平均成绩）

A A A A A A A A A A　　075
B B B B B B B B B B　　071
C C C C C C C C C C　　089
D D D D D D D D D D　　075
E E E E E E E E E E　　084
F F F F F F F F F F　　078
G G G G G G G G G G　　076
H H H H H H H H H H　　080
I I I I I I I I I I　　091
J J J J J J J J J J　　093

显示结果（全班各门课的总平均成绩）

AVERAGE=81.42

说明：

（1）从主程序读入学生成绩数据，数据文件中的每一个记录包括学生姓名、学号和五门课程成绩。本程序可以计算出人数不同的各班学生的总平均成绩。每读一个记录，就给 N 加1作学生累计数。当数据文件 STDT. DAT 中的记录读完，N 的值就是全班学生数。

（2）读入一个学生的数据后，调用子程序 CALCUL（Calculation 的缩写，表示“计算”之意）。在 CALL 语句的 USING 子句中用的参数是 STUDENT-RECORD，TOTAL-M 和 AVER-M。前者是读入学生成绩记录，它包括学生名，学号和五门课成绩。子程序 CALCUL 没有输入文件，它的数据来自主程序。联接节中组合项 STUDENT-RECORD 和主程序中记录 STUDENT-RECORD 的描述一致，在调用子程序时，就将主程序中 NAME 的值传送给子程序中的 NAMES-S，主程序中 NUMB 的值传送给子程序的 NUMB-S，等等。然后计算出 TOTAL-S，它是某一学生的五门课总分数，并计算出该学生的平均分数 AVER-S。

（3）执行完子程序返回主程序时，子程序的 TOTAL-S 的值传送给主程序的 TOTAL-M（个人总分），将 AVER-S 的值传送给主程序的 AVER-M（该学生的平均分）。然后输出该学生的姓名和平均分数。之后，使 N 加1（累计学生数）。再将每个学生的总分累加，放在 TOTAL 中。流程转回到 R 段，再读下一条记录，重复上述过程，直到文件读完为止，此时转到 C 段。将 TOTAL 除以5N（每个学生5门课），得到全班总平均分 AVERAGE。显示总平均分，关闭文件，结束。

**【例 8.7】** 计算火车费。要求用 ACCEPT 语句输入下列数据：

| 项　目 | DISTANCE<br>距　离 | TRIAN-CODE<br>火车代码 | CLASS-CODE<br>等级代码 | RESRV-CODE<br>附加费代码 | 空　格 |
|---|---|---|---|---|---|
| 数据描述<br>PIC | 9（3） | 9 | 9 | 9 | X（50） |

其中：（1）DISTANCE：表示旅行里程，以公里（km）为单位。

（2）TRIAN-CODE：当其值为0时代表慢车，1代表快车，2代表特快。

（3）CLASS-CODE：0代表普通车厢，1代表软席车厢。

（4）RESRV-CODE：0代表无附加费，1代表有附加费。

已知上述数据，要计算出相应的车费。当给出的DISTANCE（距离）＝0时，表示不再需要计算了，要求停止运行。

计算各部分费用的规定如下：

（1）基本部分（即慢车）费用：

0～50km　　2元

51～100km　　4元

超过100km的，每50km加1元。

（2）快车费：

0～100km　　3元

101～400km　　4元

超过400km的　5元

（3）特快加费：

0～100km　　4元

超过100km部分，每100km加1元。

（4）软席车厢加费：

0～100km　　5元

101～200km　　10元

201～400km　　15元

401～600km　　20元

601km以上　　25元

（5）附加费：

当此项代码＝1时，收附加费2元。

全部费用＝基本费＋加快费＋特快费＋软席费＋附加费

主程序：

```
IDENTIFICATION      DIVISION.          (标识部)
PROGRAM-ID. EX87.
ENVIRONMENT         DIVISION.          (环境部)
DATA          DIVISION.                (数据部)
WORKING-STORAGE  SECTION.
77  BASE-FEE  PIC  9 (4).             (基本费用)
77  SUPR-EXPRS-FEE  PIC  9 (4)        (特快加费)
```

```
77  EXPRESS-FEE  PIC  9 (4).          (快车加费)
77  FRST-CLASS-FEE  PIC  9 (4).       (软席加费)
77  RESRV-SEAT-FEE  PIC  9 (3).       (附加费)
77  TOTAL-FARE  PIC  9 (4).           (全部费用)
01  INPUT-DATA.
  02  DISTANCE  PIC  9 (3).           (距离)
  02  TRAIN-CODE  PIC  9.             (火车代码)
  02  CLASS-CODE  PIC  9.             (等级代码)
  02  RESRV-CODE  PIC  9.             (附加费代码)
PROCEDURE          DIVISION.          (过程部)
START-RUN.
  ACCEPT  INPUT-DATA.
  PERFORM  PROCESS-DATA
           UNTIL  DISTANCE=0.
  STOP RUN.

PROCESS-DATA.
  MOVE  0  TO  BASE-FEE, SUPR-EXPRS-FEE, EXPRESS-FEE,
               FRST-CLASS-FEE, RESRV-SEAT-FEE. (对各数据项先置初值零)
  CALL  'BASE'  USING DISTANCE, BASE-FEE.
  IF  TRAIN-CODE=1
      CALL  'EXPRESS'  USING  DISTANCE, EXPRESS-FEE
  ELSE
      IF  TRAIN-CODE=2
          CALL  'SPEXPRS'  USING  DISTANCE, SUPR-EXPRS-FEE.
  IF  CLASS-CODE=1
      CALL  'FRSCLAS'  USING  DISTANCE, FRST-CLASS-FEE.
  IF  RESRV-CODE=1
      MOVE  2  TO  RESRV-SEAT-FEE.
  COMPUTE  TOTAL-FARE =BASE-FEE+SUPR-EXPRS-FEE+EXPRESS-FEE
                       +FRST-CLASS-FEE+RESRV-SEAT-FEE.
  DISPLAY  'DISTANCE=', DISTANCE.
  DISPLAY  'BASE FEE=', BASE-FEE.
  DISPLAY  'SUPEREXPRESS  FEE=', SUPR-EXPRS-FEE.
  DISPLAY  'EXPRESS FEE=', EXPRESS-FEE.
  DISPLAY  'FIRST-CLASS FEE=', FRST-CLASS-FEE.
  DISPLAY  'RESERVED SEAT FEE=', RESRV-SEAT-FEE.
  DISPLAY  'TOTAL FARE=', TOTAL-FARE.
  ACCEPT  INPUT-DATA.
```

子程序 BASE（计算基本费）：

```
IDENTIFICATION        DIVISION.        (标识部)
PROGRAM-ID. BASE.                      (子程序名 BASE)
ENVIRONMENT           DIVISION.        (环境部)
```

```
DATA          DIVISION.                          (数据部)
WORKING-STORAGE  SECTION.                        (工作单元节)
77  W  PIC  9 (3).
LINKAGE  SECTION.                                (联接节)
77  DISTANCE  PIC  9 (3).                       (距离)
77  BASE-FEE  PIC  9 (4).                       (基本费用)
PROCEDURE    DIVISION  USING  DISTANCE, BASE-FEE.
START-BASE.
    IF  DISTANCE<51
        MOVE  2  TO  BASE-FEE.
    ELSE
          IF  DISTANCE<101
              MOVE  4  TO  BASE-FEE
          ELSE  COMPUTE  W=DISTANCE-100
                COMPUTE  BASE-FEE=4 + (W/50+1) *1.
E.   EXIT  PROGRAM.
```

子程序 SPEXPRS（计算特快加费）：

```
IDENTIFICATION          DIVISION.
PROGRAM-ID. SPEXPRS.
ENVIRONMENT             DIVISION.
DATA          DIVISION.
WORKING--STORAGE  SECTION.
77  W  PIC  9 (3).
LINKAGE  SECTION.
77  DISTANCE  PIC  9 (3).
77  SUPR EXPRS FEE  PIC  9 (4).  (特快加费)
PROCEDURE               DIVISION USING DISTANCE, SUPR-EXPRS FEE.
START-SPEXPRS.
    IF  DISTANCE<=101
        MOVE  4  TO  SUPR-EXPRS-FEE
    ELSE
         COMPUTE  W  =  DISTANCE     100
                COMPUTE  SUPR-EXPRS-FEE      4  +  (W  /  100  +  1). *1.
E.   EXIT  PROGRAM.
```

子程序 EXPRESS（计算加快费）：

```
IDENTIFICATION          DIVISION.
PROGRAM ID, EXPRESS.
ENVIRONMENT             DIVISION.
DATA          DIVISION.
WORKING-STORAGE     SECTION.
77  W  PIC  9 (3).
LINKAGE                 SECTION.
```

```
77  DISTANCE  PIC  9 (3).
77  EXPRESS-FEE  PIC  9 (4). (快车加费)
PROCEDURE              DIVISION USING DISTANCE, EXPRESS-FEE.
START-EXPRESS.
    IF  DISTANCE  <  101
        MOVE  3  TO  EXPRESS-FEE
    ELSE
        IF  DISTANCE <  401
            MOVE  4  TO  EXPRESS-FEE
        ELSE
            MOVE  5  TO  EXPRESS-FEE.
E.   EXIT  PROGRAM.
```

子程序 FRSCLAS（计算软席费）：

```
IDENTIFICATION         DIVISION.
PROGRAM-ID, FRSCLAS.
ENVIRONMENT            DIVISION.
DATA                   DIVISION.
LINKAGE                SECTION.
77  DISTANCE  PIC  9 (3).
77  FRST-CLASS-FEE  PIC  9 (4). (软席加费)
PROCEDURE              DIVISION USING DISTANCE, FRST-CLASS-FEE.
START-FRSCLAS.
    IF  DISTANCE  <  101
        MOVE  5  TO  FRST-CLASS-FEE
    ELSE
        IF  DISTANCE  <  201
            MOVE  10  TO  FRST-CLASS-FEE
        ELSE
            IF  DISTANCE  <  401
                MOVE  15  TO  FRST-CLASS-FEE
            ELSE
                IF  DISTANCE  <  601
                    MOVE  20  TO  FRST-CLASS-FEE
                ELSE
                    MOVE  25  TO  FRST-CLASS-FEE.
E.   EXIT  PROGRAM.
```

说明：

(1) 这个题目仅为练习用，计算费用的方法和给出的具体价格都是任意给的，仅为说明分项计算任务可以用不同的子程序来完成。

(2)有的计算机系统的COBOL 要求CALL 语句中子程序名要用引号括起来，如CALL 'Λ'（有的计算机系统不要求)。有的系统要求在输入源程序时在主程序和子程序结束的最后一行使用“END PROGRAM 程序名”（例如“END PROGRAM EX86”，“END PRO-

GRAM BASE" 等)，有的则不要求。使用时请注意所使用的系统的规定。

(3) 此程序用 ACCPECT 输入和 DISPLAY 输出，读者可以修改此程序，用 READ 读入数据，用 WRITE 输出数据。

(4) 请读者自己看懂这个程序，并画出主程序和子程序的流程图，请特别注意 IF 语句的嵌套关系。

## 习　　题

8.1　子程序包括哪几部分?它和主程序有什么共同和不同之处?

8.2　主程序和子程序是怎样发生数据联系的?在用 USING 子句时要遵守什么规定?

8.3　联接节 (LINKAGE SECTION) 的作用是什么?什么样的数据项应在联接节中描述?

8.4　将本章例8.7程序的总框图和各子程序的框图画出来。

8.5　将第六章的例6.2程序改写，把"处理输入记录"部分功能编写成一个子程序，然后用一个主程序调用它。

# 第九章　表的建立和查找

## §9.1　表 的 概 念

COBOL 语言中的表（table）大体相当于其它高级语言中的数组（array），但也不是完全相同，以后将会看到有一些区别。所谓“表的建立”就是定义一个“表”，在其它语言中叫做“定义一个数组”。定义（建立）一个“表”后，就可以对它所包含的各个元素（称“表元素”，相当于“数组元素”）赋值并引用。这些合起来称作“表处理”（Table handling）。

下面具体介绍一下 COBOL 语言中的“表”。

到目前为止，我们用一个数据名代表一个数据项，如果有多个数据项，就要用多个数据名来代表。有时，有一些数据项的地位和作用是相同的，数据描述也是相同的，但还要一个一个地分别定名和描述。如：有一个居民记录，内包含几个居民的号码、性别、年龄、地址等数据。需要为每一居民单独起一名字，并分别描述。如：

```
01  PERSONAL-RECORD.
  02  ZHANG.
    03  NUM PIC  9 (4).
    03  SEX  PIC  X.
    03  AGE  PIC  9 (2).
    03  ADDR  PIC  X (13).
  02  LI.
    03  NUM  PIC  9 (4).
    03  SEX  PIC  X.
    03  AGE  PIC  9 (2).
    03  ADDR  PIC  X (13).
  02  WANG.
      ⋮
  02  FUN.
      ⋮
```

其在内存中的存储情况见图9.1。

| PERSONAL-RECORD | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ZHANG | | | | LI | | | | WANG | | | | FUN | | | |
| NUM | SEX | AGE | ADDR | NUM | SEX | AGE | ADDR | NUM | SEX | AGE | ADDR | NUM | SEX | AGE | ADDR |
| 9(4) | X | 9(2) | X(13) | 9(4) | X | 9(2) | X(13) | 9(4) | X | 9(2) | X(13) | 9(4) | X | 9(2) | X(13) |

图　9.1

显然这是很不方便的，如果有100个居民，那么数据部中记录的描述将会很长。而且要定

义100个名字，引用起来也很不方便。例如需要求这100个居民的平均年龄，则要写成：

```
ADD AGE OF ZHANG,AGE OF LI,AGE OF WANG,… TO AGE OF LING.
```

（"…"表示其它96个居民的年龄，为节约篇幅，省写）

即将前99个人的年龄加到第100个人的年龄上去（假设最后一人姓名为 LING）。或者分开写：

```
ADD AGE OF ZHANG TO AGE OF LI.
ADD AGE OF LI    TO AGE OF WANG.
  ⋮                 ⋮
ADD AGE OF FUN   TO AGE OF LING.
```

即将第一、二个相加，再加上第三个，……直至加到第100个，这更麻烦。当然还有其它的办法进行相加，但都是比较麻烦的。全部相加完后才能求平均年龄。

在管理工作中，类似的问题是很多的，例如一个班有26个学生，要登记每个学生的学号和总成绩，见图9.2。

某 班 学 生 成 绩 记 录

| 学生 A | | 学生 B | | …… | 学生 Y | | 学生 Z | |
|---|---|---|---|---|---|---|---|---|
| 学 号 | 成 绩 | 学 号 | 成 绩 | …… | 学 号 | 成 绩 | 学 号 | 成 绩 |
| 9(4) | 9(3)V9 | 9(4) | 9(3)V9 | …… | 9(4) | 9(3)V9 | 9(4) | 9(3)V9 |

图 9.2

又如工厂中产品的生产—销售—库存记录，见图9.3。

产品生产—销售—库存记录

| 产 品 1 | | | 产 品 2 | | | …… | 产 品 19 | | | 产 品 20 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 产量 | 销售量 | 库存量 | 产量 | 销售量 | 库存量 | …… | 产量 | 销售量 | 库存量 | 产量 | 销售量 | 库存量 |
| 9(6) | 9(6) | 9(6) | 9(6) | 9(6) | 9(6) | | 9(6) | 9(6) | 9(6) | 9(6) | 9(6) | 9(6) |

图 9.3

它的数据描述为：

```
01  PRODUCT-RECORD.
  02  PRODUCT-1.
    04  QUANTITY-OF-PRODUCTION  PIC  9 (6). (生产数)
    04  QUANTITY-OF-SALES  PIC  9 (6). (销售数)
    04  QUANTITY-ON-HAND  PIC  9 (6). (库存数)
  02  PRODUCT-2.
    04  QUANTITY-OF-PRODUCTION  PIC  9 (6).
    04  QUANTITY-OF-SALES  PIC  9 (6).
    04  QUANTITY-ON-HAND  PIC  9 (6).
```

```
    ⋮
02  PRODUCT-20.
    ⋮
```

为解决这种重复性的工作，我们可以用一个统一的名字去代表那些地位、作用和描述相同的数据项。或者说，把具有相同属性的数据项按一定的逻辑顺序组织在一起，成为一个整体的数据组织，用一个统一的名字来代表它们，这就是“表”。构成“表”的各数据项称为“表元素”。“表”和“表元素”大体上相当于其它语言中“数组”和“数组元素”的概念。在建立一个“表”以后，只要指出表名和序号（即指出表中第几个元素）或表中的相对地址位置（相对于表中第一个元素的字节地址），就可唯一地确定一个表元素。序号称为“下标”，相对地址位置称为“位标”，“下标”和“位标”统称为“出现号”（occurs number）。如上面的产品记录，我们可以将20个产品的数据项组成一个“表”，以PRODUCT作表名。以PRODUCT（1）表示这个“表”中的第一个元素，即第一个产品的数据项，以PRODUCT（N）表示第N个产品的数据项。括弧中的1和N代表某一表元素在表中的序号，它们是“下标”（subcript）。这里是用“下标”作为“出现号”。在本章§9.7中，可以看到除了“下标”以外还可以用“位标”作“出现号”来引用某一个表元素。

数据处理中常常不是以数学式子说明问题，而以各种数据表格说明问题。因此，COBOL没有引入许多纯数学的名词（包括数组）。一个“表”的结构是有规律排列的一张表格。如图9.4所示。

PRODUCT-RECORD（产品记录）

| PRODUCT（1） | | | PRODUCT（2） | | |
|---|---|---|---|---|---|
| QUANTITY（1） | QUANTITY（2） | QUANTITY（3） | QUANTITY（1） | QUANTITY（2） | QUANTITY（3） |
| 9（6） | 9（6） | 9（6） | 9（6） | 9（6） | 9（6） |

PRODUCT-RECORD（产品记录）

| PRODUCT（3） | | | …… | PRODUCT（20） | | |
|---|---|---|---|---|---|---|
| QUANTITY（1） | QUANTITY（2） | QUANTITY（3） | | QUANTITY（1） | QUANTITY（2） | QUANTITY（3） |
| 9（6） | 9（6） | 9（6） | | 9（6） | 9（6） | 9（6） |

图 9.4

（注：由于页面宽度不够，故将PRODUCT-RECORD记录分成上下两栏印出。应当是PRODUCT-RECORD下属PRODUCT（1），PRODUCT（2），PRODUCT（3），…PRODUCT（20））

这就是“表”这个名字的来历。

## §9.2 表的建立

如前所述，表的建立就是定义一个“表”（定义一个数组）。

如果一个数据名不是代表单一的数据项，而是一个“表”的名字，则应在数据部中作说明，指出这个“表”包含多少个表元素。如：

```
01  STUDENT-RECORD.
    02  NAME PIC X (20).
    02  COURSE OCCURS 5 TIMES PIC 9 (3).
```

上面是一个学生成绩的记录，一个学生考5门课，要记录5门课的成绩。数据名COURSE表示“课程”。OCCURS 是“出现”的意思。OCCURS 5 TIMES 就是“出现5次”，或者说“重现5次”。即指出有5个 COURSE（课程）。它的数据结构为：

STUDENT-RECORD（学生记录）

| NAME（名字） | COURSE（1） | COURSE（2） | COURSE（3） | COURSE（4） | COURSE（5） |
|---|---|---|---|---|---|
| X（20） | 9（3） | 9（3） | 9（3） | 9（3） | 9（3） |

COURSE 就是“表”名。这个“表”包含五个同样类型和地位的数据项。定义这个“表”时只用了一个 OCCURS 子句，它是一维表。

表元素可以是初等项，也可以是组合项。如对图9.3所示的工厂产品记录，可以将20个产品的数据项组成一个“表”。

```
01  PRODUCT-RECORD.
    03  PRODUCT OCCURS 20 TIMES.
        04  QUANTITY-OF-PRODUCTION PIC 9(6).
        04  QUANTITY-OF-SALES PIC 9(6).
        04  QUANTITY-ON-HAND PIC 9(6).
```

上面的 PRODUCT 是“表”的名字，它包含20个“表元素”，每一个表元素都是组合项。它的数据结构如图9.5所示。

PRODUCT-RECORD

| PRODUCT（1） | | | PRODUCT（2） | | |
|---|---|---|---|---|---|
| QUANTITY—OF—PRODUCT | QUANTITY—OF—SALES | QUANTITY—ON—HAND | QUANTITY—OF—PRODUCT | QUANTITY—OF—SALES | QUANTITY—ON—HAND |
| 9（6） | 9（6） | 9（6） | 9（6） | 9（6） | 9（6） |

| PRODUCT-RECORD | | | |
|---|---|---|---|
| …… | PRODUCT (20) | | |
| …… | QUANTITY OF PRODUCT | QUANTITY OF SALES | QUANTITY ON HAND |
| …… | 9 (6) | 9 (6) | 9 (6) |

图 9.5

每一个 PRODUCT 元素包含三个初等项。

我们进一步看，三个初等项都是“数量”(生产数量，销售数量，库存数量)。数据描述都是9 (6)。它们的地位和描述又都相同，因此可以进一步简化，写成：

```
01  PRODUCT-RECORD.
  03  PRODUCT OCCURS 20 TIMES.
    04  QUANTITY OCCURS 3 TIMES PIC 9(6).
```

即在表 PRODUCT 中的每一个表元素下面，又建立一个“表” QUANTITY，包括三个 QUANTITY 元素，每个 QUANTITY 元素都是初等项。它的数据结构见图9.4。如果要找第三种产品的销售数量，要用两个下标才能确定，即 QUANTITY (3, 2)。第一个下标是指出第几个产品，第二个下标指出第几个数量。因此，QUANTITY 是一个二维表。有的初学者往往容易搞混，把 PRODUCT 误认为是二维表。我们引用一个 PRODUCT 的元素，只需要用一个下标，如 PRODUCT (3) 是唯一的，它是一个包含其下属的 QUANTIAY (1)，QUANTITY (2)，QUANTITY (3) 的组合项。因此，PRODUCT 是一维表，QNANTITY 是二维表。因为用 QUANTITY (1) 不能唯一确定一个表元素，而用 QUANTITY (3, 1) 能唯一确定一个表元素。简单地说，引用表元素时只需指定一个下标的，是一维表。引用表元素时需指定两个下标的是二维表。

可以直接从数据部的描述中看出：如果在数据项描述体中有一个 OCCURS 子句，而在它的上属数据项的描述体中没有 OCCURS 子句，则它是一维表。如果在它的上属项之中有一个描述体中又带有 OCCURS 子句，则它为二维表。上例中的 QUANTITY 本身带 OCCURS 子句，其上属项 PRODUCT 又带 OCCURS 子句，所以 QUANTITY 是一个二维表。它有20×3=60元素。引用二维表元素时，每一个下标指出该维中元素的序号。例如 QUANTITY (1, 2) 表示 QUANTITY 表中第一维中序号为1、第二维中序号为2的元素，即相当于数组中第一行、第二列的元素。

可以定义三维表。如果有十个工厂，每个工厂各有20个产品，每个产品都要统计上述三种数量，则可以定义一个三维表，在上面二维表的描述基础上再增加一个02层。

```
01  PRODUCT RECORD.
  02  FACTORY OCCURS 10 TIMES.
    03  PRODUCT OCCURS 20 TIMES.
      04  QUANTITY OCCURS 3 TIMES PIC 9(6).
```

如果要找第五个工厂第三个产品第二种数量（销售数量）则需要用三个下标，即QUANTITY (5, 3, 2)。整个PRODUCT-RECORD中包含十个FACTORY项，每个FACTORY项中又包含20个PRODUCT项，每个PRODUCT又包含三个QUANTITY初等项。它共有10×20×3=600个QUANTITY元素。长度为600×6=3600字节。请读者自己画出它的数据的层次结构。有些读者可能想，一个记录能容纳这么多的字符吗？从磁盘或磁带输入的一个记录可以容纳很多字符。譬如IBM2311磁盘系统一个记录最多可容纳3605个字符（一个字符占一个字节）。在内存中记录区的长度可以更大，例如M150F中可达13万字节。

为了帮助读者进一步弄清楚多维表的概念。下面再举一个例子来分析。

```
01  TABLE.
  02  A1 OCCURS 5 TIMES.
    03  B1 PIC X(3).
    03  B2 OCCURS 10 TIMES PIC 9(4).
    03  B3 OCCURS 8 TIMES.
      04  C1 PIC 9(6).
      04  C2 OCCURS 20 TIMES PIC 99.
      04  C3 PIC 9(4).
    03  B4 PIC X(3).
  02  A2 PIC X(5).
  02  A3 PIC X(10).
```

其中：(1)A1，B1，B4是一维表，因为它们本身或其上属项的描述中只出现了一次OCCURS子句。引用这些表中的元素只需指定一个下标。如A1(3)，B1(5)，B4(1)等。写在括号中的可以是数值常量也可以是具有整数值的数据项名。

(2)B2，B3，C1，C3是二维表。因为它们本身和其上属项的描述中有两个OCCURS子句，引用它们需要用两个下标。如：B2(5,3)，B3(4,8)，C1(2,7)，C3(1,8)等。

(3)C2是一个三维表，因为在它本身和其上属项的描述中一共出现了三个OCCURS子句。要引用它的元素，要用三个下标，如C2(4,3,1)，即括弧内应有三个“出现号”。

(4) A2，A3不是“表名”，而是普通的初等数据项。

OCCURS子句的最简单的格式为：

```
OCCURS  整数  TIMES
```

以后还会介绍OCCURS子句的其它格式。

关于在数据部中定义“表”的有关说明：

(1) OCCURS后面只能为正整数，如OCCURS 4合法，OCCURS 3.5，OCCURS 0非法。

(2) OCCURS子句不能出现在77层，因为77层是独立的数据项，不和其它数据发生组合关系的，下面用法是错误的：

```
77  A OCCURS 5 PIC X(3).
```

(3) OCCURS子句不能用于01层，例如下面用法是错误的：

```
01  IN-REC OCCURS 4 TIMES.
  03  PRODUCT-NUM PIC 9(4).
```

```
    03  PRODUCT-NAME PIC X(16).
```

“表”只能定义在记录内部，如果想在一个记录中包含几个产品的数据，可以在01层下面增加一层（例如02层），把表定义在这一层上。如：

```
01  IN-REC.
  02  PRODUCT OCCURS 4 TIMES.
    03  PRODUCT-NUM  PIC 9(4).
    03  PRODUCT NAME  PIC X(16).
```

在提供的记录中，每20字节为一组产品的数据，一个记录包含四个产品的数据。

(4) 如果用OCCURS来描述一个初等项，则“重复出现”的是这个初等项。如果用OCCURS来描述一个组合项，则“重复出现”的是这个组合项。

(5) 在COBOL中表元素可以允许为组合项。而且在建立一个多维表的同时，也建立了一维（或一、二维）表。这是和其它语言中的数组不同之处。如：

```
01  A.
  02  B OCCURS 10.
    04  C OCCURS 5.
      05  D OCCURS 8 PIC X(10).
```

建立了三维表D，同时建立了二维表C，一维表B，引用时可以任意引用一维的，二维的或三维的表的元素，例如B (2)，C (2，1)，D (2，1，5)，请读者分析其中的含义。

(6) ANSI COBOL 1974允许最多用到三维表。有的计算机系统配备的COBOL级别比较低，可能只能用二维甚至一维，用时应查说明书。

(7) 只有当OCCURS所说明的数据是初等项时，才能在该数据项的描述中使用PIC子句。如上面定义的三维表中，只能在05层才能用PIC子句，在该组合项中，05层是最低的层次，它下面没有从属项，而

```
01  A.
  02  B OCCURS 5 PIC X(30).
    03  C PIC X(20).
    03  D PIC 9(10).
```

是错误的。02层下面还有03层下属项。而PIC子句不能用来描述组合项。

(8) 不能用VALUE子句对表赋初值。不能同时用OCCURS子句和VALUE子句来描述同一个数据项。因为用VALUE子句只能对一个数据项赋初值，而不能对所有表元素赋值。例如下面用法是错的。

```
02  UNIT-PRICE OCCURS 5 PIC 9(3)V99 VALUE 500.
```

在带OCCURS子句的数据项的下属项的描述体中也不能出现VALUE子句。如：

```
02  A OCCURS 5.
  03  A1 PIC 9(3) VALUE 100.
  03  A2 PIC 9(2) VALUE 10.
```

也是错的。因为A1和A2都是表名，不是一个普通的数据项，而VALUE子句是不能用来对表赋初值的。

(9) 多维表的元素在内存中是以行排列的。如果A是一个二维表，每一维中包含两个元素，在内存中按以下顺序排列：A(1,1)⇒A(1,2)⇒A(2,1)⇒A(2,2)。如果A是三维表，每

一维有三个元素，则排列的顺序为：A(1,1,1)⇒A(1,1,2)⇒A(1,1,3)⇒A(1,2,1)⇒A(1,2,2)⇒A(1,2,3)⇒A(1,3,1)⇒A(1,3,2)⇒A(1,3,3)⇒A(2,1,1)⇒A(2,1,2)⇒A(2,1,3)⇒A(2,2,1)⇒A(2,2,2)⇒A(2,2,3)⇒…

用“表”来组织同属性的数据项的优点在于：

(1) 以一个名字统一代表它们，表示它们是相同地位的。各数据项间关系明确。

(2) 可以不必为每一数据项分别起名，只要指出“表”名和“出现号”(下标或位标)就能唯一地指出每一个表元素，使程序简单，使用方便。

(3) 由于是用“出现号”(今用“下标”作“出现号”)来指出表元素在表中的位置，因此特别有利于用循环方法来处理。只要每次改变下标的数值，就可用同一段程序重复处理不同的表元素。

(4) 当表包括的元素个数需变化时，只需改变数据部中的OCCURS子句，而程序的过程部可以不必改变。例如，如果原来处理50个表元素，今要改为处理100个表元素，只需将OCCURS 50改成OCCRUS 100即可，使用灵活。

## §9.3 可变长表

有时一个“表”中需要包含多少个表元素并不是固定的，例如，一个学生的成绩记录中每个学生选读的课程门数不同，每门课程都要分别记录成绩，见下表。

| STUDENT-SCORE-RECORD | | | | | | |
|---|---|---|---|---|---|---|
| NAME | QTY-OF COURSE | SCORE(1) | SCORE(2) | SCORE(3) | …… | SCORE(N) |
| X(10) | 9(2) | 9(3) | 9(3) | 9(3) | | 9(3) |

在学生名字这一项后面，有一项是“课程数”，如果它为8，则后面应记录8项成绩。如果选5门课，则应记录5项成绩。显然，不同学生的记录长度是不同的，表的元素个数也是不同的。

可以用下面形式的OCCURS子句

```
01  STUDENT-SCORE-RECORD.
  03  NAME PIC X(10).
  03  QTY-OF-COURSE PIC 9(2).
  03  SCORE OCCURS 1 TO 15 TIMES
        DEPENDING ON  QTY-OF-COURSE PIC 9(3).
```

一般格式为：

OCCURS 整数1 TO 整数2 TIMES
DEPENDING ON 数据名1

数据名1(如上例中的QTY-OF-COURSE)可以在表所在的记录中描述。也可以不在本记录中描述，例如可以是工作单元节中的一个77层的数据项。例如：

```
DATA    DIVISION.
FILE    SECTION.
FD  IN-FILE  LABEL RECORD  IS  STANDARD.
01  STUDENT-SCORE-RECORD.
  03  NAME  PIC X(10).
  03  SCORE  OCCURS  1  TO  15  TIMES
        DEPENDING ON QTY-OF-COURSE  PIC 9(3).
WORK IN G-STORAGE  SECTION.
77  QTY-OF-COURSE  PIC  9(2).
    ⋮
PROCEDURE  DIVISION.
    ⋮
    ACCEPT    QTY-OF-COURSE.
    READ  IN-FILE.
    ⋮
```

从键盘输入 QTY-OF-COURSE 的值，如果输入的值为4，则读入的 STUDENT-SCORE-RECORD 记录应包含：姓名（NAME）和四门课程的成绩（SCORE (1)，SCORE (2)，SCORE (3)，SCORE (4)）。

应注意：QTY-OF-COURSE 不能是“表”中的一部分，如果写成下面这样是错误的。

```
03  SCORE OCCURS 1 TO 15
      DEPENDING ON QTY-OF-COURSE.
  04  QTY-OF-COURSE  PIC  9(2).
  04  SCORE-OF-COURSE  PIC  9(3).
```

因 QTY-OF-COURSE 是 SCORE 表中的一部分。

可变长表在经济管理中是有用的。例如，顾客来往帐目的记录，每来往一笔交易就登记一次，如果一个顾客一个月中有100笔交易，而另一顾客只有5次交易。显然，二者的记录长度是不同的。在磁带或磁盘上允许建立不同长度的记录。

## §9.4 表元素的引用

“表”必须先在数据部定义，才能在过程部引用。即先定义，后引用。

各个表元素是按一定规律在内存中顺序存放的，因此，在引用时必须指出表名和这个元素在表中的位置。例如 PRODUCT (2)。

如果是二维表，在括弧中要用两个下标分别表示此元素在每一维中的位置。如 QUANTITY (3, 2)。在第一个下标和第二个下标之间用一个以上空格隔开，或用一个逗号后跟一个空格分隔。在括弧的外侧要留空格，内侧不必留空格。

注意：(一) 如果已说明 B 是一个表。

```
01  TABLE.
  02  B OCCURS  8.
    03  C1  PIC  X(8).
    03  C2  PIC  X(2).
```

不能直接引用表名B而不加下标。如：

```
MOVE  B  TO  A.
```

是错误的。因为这个表中有8个B的元素，你要的是哪一个B呢?必须用表名加下标（用括弧将下标括起来）来指定某一个表元素，如B（1）或B（5）等。如果想引用的是B的全部表元素，则应写：

```
MOVE  TABLE  TO  A.
```

从图9.6可以看到，在内存中只有TABLE组合项，即B（1）、…，B（8）组合项，而不存在一个B项，因而无法引用它。

| TABLE | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| B(1) | | B(2) | | B(3) | | B(4) | | B(5) | | B(6) | | B(7) | | B(8) | |
| C1 | C2 | C1 | C2 | C1 | C2 | C1 | C2 | C1 | C2 | C1 | C2 | C1 | C2 | C1 | C2 |
| X(8) | X(2) | X(8) | X(2) | X(8) | X(2) | X(8) | X(2) | X(8) | X(2) | X(8) | X(2) | X(8) | X(2) | X(8) | X(2) |

图 9.6

（二）如果表元素是组合项，则引用它下属的项（可以是初等项或组合项），也必须用下标指明它是属于哪一个表元素的。如上例中，写：

```
MOVE  C1  TO  A.
```

是错误的。因为从图9.6可以看出有多个C1和C2，你要的是哪一个C1呢?应写成：

```
MOVE  C1  (2)  TO  A.
```

表示是送B(2)中的C1到A去。

（三）如果表元素是组合项，可以用它对下属的数据项进行限定。如上面已定义了一个B表，在过程部中可以写：

```
MOVE  SPACE  TO  C1  OF  B (2).
```

表示是B(2)中的C1。但带下标的数据名不能被限定。如：

```
MOVE  SPACE  TO  B (1)  OF  TABLE.
```

或：

```
MOVE  10  TO  QUANTITY  (2)  IN  PRODUCT  (3).
```

都是不对的。也就是说，表元素应出现在限定词OF或IN的后面，而不能在它们的前面。分析下面用法：

| | | | | |
|---|---|---|---|---|
| 数据名 OF (IN) | 表元素. 如C2 | OF B (3) | 形式 | (可以) |
| 表元素 OF (IN) | 数据名. 如B (2) | OF T | 形式 | (不可以) |
| 表元素 OF (IN) | 表元素. 如D (1) | OF C (1) | 形式 | (不可以) |

（四）下标只能是整常数或具有整型值的数据名，如A（I），A（3）为合法，不能是表达式，如：A（I+1），A（3+5）不合法。

下标如为数据名，则此数据名可以在数据部中的文件节或工作单元节定义，也可在表所在的记录中定义（描述），但不能属于“表”当中的一部分。

如：

```
01  A.
    02  B  OCCURS  4.
        03  C1  PIC  9(2).
        03  C2  PIC  9(3).
    02  D       PIC  9(4).
```

引用表元素时，B (D)为合法，而B(C1)不合法。

(五) 标准 COBOL 规定，下标不能是带下标的数据名，即不能是表元素。如 A (A (1)) 或 A (B (2)) 非法。

(六) 带下标的数据名(表元素)又称下标数据名(相当于其它语言中的“下标变量”)。它是标识符的一种形式。

## §9.5 给表元素赋初值

前面已说明，不能在一个数据项描述体中同时用 OCCURS 子句和 VALUE 子句给各个表元素赋初值。当然人们可以在过程部用 READ，MOVE 以及 ADD，COMPUTE 等算术运算语句给各表元素赋值，但是这要占运行时间。往往有这样的程序，每次运行之前，都要求表元素具有某些确定的初值。如果能在程序编译时就已使这些表元素中有了确定的值，则可以省去每次运行开始时用过程部语句传送赋值的时间，这对某些需要多次运行的程序尤为重要。这样，就可以把源程序编译、联接后得到的目标程序存放在磁盘上，需要时直接调出目标程序运行即可，此时已有了初值了。

在工作单元节中对表元素赋初值的办法有二：

第一种方法是：对包括所有表元素的整个表赋给一个初值。这时可以对表的描述体上面一层的数据项赋一个初值即可。例如，有一个表 A：

```
01  TABLE.
    03  A  OCCURS  20  PIC  9(3).
```

我们想使 A 表的全部元素初值为零，不能这样写：

```
03  A  OCCURS  20  PIC  9(3)  VALUE  IS  ZEROS.
```

可以在01层中用 VALUE 子句对 TABLE 赋以一个初值即可。

```
01  TABLE  VALUE  IS  ZERO.
    03  A  OCCURS  20  PIC  9(3).
```

TABLE 是组合项，使它的值为零，就相当于每个 A 元素中全放满了零字符，等于 A 各元素的值都是零。

也可以这样用：

```
01  T  VALUE'ABCDEFHIJ'.
    02  Q  OCCURS  3  PIC  X(3).
```

这样，表元素 Q(1)，Q(2)，Q(3)的内容分别是'ABC'，'DEF'，'HIJ'.

但是当表中包含元素较多，而且每个元素长度又较大 (例如20个表元素每个元素含20个字符)，用这样办法给表元素赋初值就困难了。因 COBOL 规定的字符串长度是有限的，不能在 VALUE 子句中写很长的非数值常量。

第二种方法是：联合使用 OCCURS 子句和 REDEFINES 子句来给各个表元素赋初值，

这种方法比较巧妙。它的步骤是：(1) 先在工作单元节中定义一个组合项，它占的内存的大小和需赋值的表一样，在该组合项中定义若干个数据项，数据项的描述和表的元素相同。(2) 然后对这些数据项分别用 VALUE 子句赋以初值，由于在这些数据项的描述中没有出现 OCCURS 子句，因此用 VALUE 赋初值是合法的。这些值就是要赋给表元素的初值。(3) 把这个组合项重定义成一个表，这样就相当于对这些表元素赋了初值。

例如，想对产品单价赋初值如下（今为简单起见，只对五个产品赋初值）：

| 产品号 | 单　价 |
|---|---|
| 0010 | 000200 |
| 0020 | 000500 |
| 0030 | 000300 |
| 0040 | 000250 |
| 0050 | 005000 |

我们先定义一个组合项 UNIT PRICE-TABLE，再把表 TABLE 定义在同一段内存区上。

```
01  UNIT-PRICE-TABLE.
  02  FILLER PIC X(10) VALUE '0010000200'.
  02  FILLER PIC X(10) VALUE '0020000500'.
  02  FILLER PIC X(10) VALUE '0030000300'.
  02  FILLER PIC X(10) VALUE '0040000250'.
  02  FILLER PIC X(10) VALUE '0050005000'.
01  UNIT-PRICE TABLE-R REDEFINES UNIT-PRICE-TABLE.
  02  TABLE OCCURS 5.
    03  PROD-CODE PIC 9(4).
    03  UNIT-PRICE PIC 9(6).
```

在内存中，两个组合项 UNIT-PRICE-TABLE 和 UNIT-PRICE TABLE R 共享同一区域的内容，见图9.7所示。

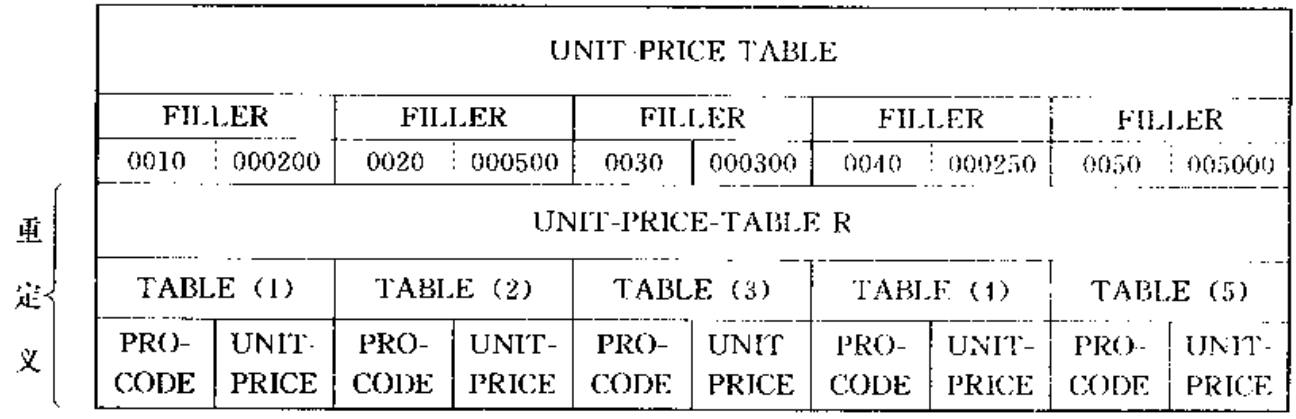



图　9.7

可以看出：UNIT-PRICE-TABLE 中的每一个数据项(FILLER)分别与 UNIT-PRICE-TABLE-R 中的 TABLE 表的每一个元素同占一个内存区域。而在每一个 TABLE 的元素中，PRO-CODE 为4个字节，UNIT-PRICE 占6个字节，因此，在第一个 FILLER 中的10个字符与 TABLE (1) 中的 PRO-CODE 和 UNIT-PRICE 共享。也就是说，PRO-CODE (1)

得到初值0010，UNIT-PRICE (1)得到初值000200。PRO-CODE (2) 得到初值0020，UNIT-PRICE (2) 得初值000500，余类推。这就解决了对表元素分别赋初值的问题。

有的读者可能不解：为什么要绕了一个圈子来给表元素赋初值呢?这是因为 VALUE 子句和 OCCURS 子句不能同时用来描述一个数据项。因此，先设法在内存中开辟一个区域，内设若干数据项，在其中先放入必要的数据，然后使各个表元素分别与该区域中各数据项对应，这样，同一个内存区有了两个名字，均可被引用了，事实上，需要引用的只是表元素，因此，在事先设立的组合项 (UNIT-PRICE-TABLE) 中各数据项名均取 FILLER，其作用无非占一块内存以便放数据而已。(当然，也可以不用 FILLER 而用任何其它名字，如 A，B，C，D……等。)

## §9.6 表的应用举例

**【例 9.1】** 有一批货物，其货号和单价如下：

| 货号 | 单 价 | 货号 | 单 价 |
|---|---|---|---|
| 0010 | 000200 | 0130 | 000650 |
| 0020 | 000500 | 0150 | 001670 |
| 0030 | 000300 | 0190 | 003000 |
| 0040 | 000250 | 0200 | 000127 |
| 0050 | 005000 | 0230 | 000098 |
| 0060 | 000450 | 0240 | 000160 |
| 0090 | 000125 | 0270 | 000320 |
| 0100 | 000670 | 0290 | 000470 |
| 0110 | 001200 | 0300 | 000290 |
| 0120 | 000390 | 0330 | 000960 |

今输入一批顾客购货的记录（包括日期、货号、顾客名、购货数量)。要计算机从单价表中查出该货号产品的单价，算出顾客购货所需款项（即商店销售款)，在磁盘上另建立一个新文件，其记录中除包括输入记录中各项外再增加该货号产品的单价和销售款数。

题目分析：每次输入一货号，都要查单价，由于货物种类不多（20种)，所以可以把单价表放在内存中，以能迅速查找。设一个表 TABLE，它包括20个产品表元素，每个元素中再包括货号和单价，即：

| TABLE (1) | | TABLE (2) | | TABLE (3) | | …… | TABLE (20) | |
|---|---|---|---|---|---|---|---|---|
| PRO -CODE | UNIT -PRICE | PRO -CODE | UNIT -PRICE | PRO -CODE | UNIT -PRICE | …… | PRO -CODE | UNIT -PRICE |
| 9 (4) | 9 (6) | 9 (4) | 9 (6) | 9 (4) | 9 (6) | …… | 9 (4) | 9 (6) |

用上面介绍的方法，对每一个货号 (PRO-CODE) 和单价 (UNIT-PRICE) 赋以初值，以后每输入一个货号，就逐个地和各个表元素对比，找出该货号，查出单价。再将“单价×

数量”得销售款数，问题就解决了。

程序为：

```
IDENTIFICATION          DIVISION.
PROGRAM-ID. EXAM91.
ENVIRONMENT             DIVISION.
INPUT-OUTPUT      SECTION.
FILE CONTROL.
    SELECT  OLD-SALES-FILE  ASSIGN  TO  IN-FILE.
    SELECT  NEW-SALES-FILE  ASSIGN  TO  P-FILE.
DATA      DIVISION.
FILE      SECTION.
FD    OLD-SALES-FILE  LABEL  RECORD  IS  STANDARD
      DATA  RECORDS  ARE  OLD-SALES-RECORD.
01  OLD-SALES-RECORD.
    02  YR -MON-DAY-O.
          03  YEAR     PIC  99.
          03  MONTH    PIC  99.
          03  DAYY     PIC  99.
    02  PRODUCT-CODE     PIC  9(4).
    02  CUSTOMER-CODE    PIC  9(4).
    02  QUANTITY         PIC  9(6).
FD  NEW-SALES FILE  LABEL  RECORD  IS  STANDARD
      DATA  RECORD  IS  NEW-SALES-RECORD.
01  NEW-SALES-RECORD.
    02  FILLER  PIC  X.
    02  YR-MON-DAY N.
        03  YEAR-N       PIC  99.
        03  MONTH-N      PIC  99.
        03  DAYY-N       PIC  99.
    02  PRODUCT-CODE-N       PIC  9(4).
    02  CUSTOMER-CODE-N      PIC  9(4).
    02  QUANTITY-N           PIC  9(6).
    02  UNIT-PRICE-N         PIC  9(6).
    02  SALES-VALUE-N        PIC  9(8).
WORKIN9-STORAGE  SECTION.
77  I  PIC  99. (I 用来作表的下标)
01  UNIT-PRICE-TABLE.
```

```
    02  FILLER  PIC  X(10)  VALUE  '0010000200'.
    02  FILLER  PIC  X(10)  VALUE  '0020000500'.
    02  FILLER  PIC  X(10)  VALUE  '0030000300'.
    02  FILLER  PIC  X(10)  VALUE  '0040000250'.
    02  FILLER  PIC  X(10)  VALUE  '0050005000'.
    02  FILLER  PIC  X(10)  VALUE  '0060000450'.
    02  FILLER  PIC  X(10)  VALUE  '0090000125'.
    02  FILLER  PIC  X(10)  VALUE  '0100000670'.
    02  FILLER  PIC  X(10)  VALUE  '0110001200'.
    02  FILLER  PIC  X(10)  VALUE  '0120000390'.   "产品代码——单价对照表"
    02  FILLER  PIC  X(10)  VALUE  '0130000650'.   的数据
    02  FILLER  PIC  X(10)  VALUE  '0150001670'.
    02  FILLER  PIC  X(10)  VALUE  '0190003000'.
    02  FILLER  PIC  X(10)  VALUE  '0200000127'.
    02  FILLER  PIC  X(10)  VALUE  '0230000098'.
    02  FILLER  PIC  X(10)  VALUE  '0240000160'.
    02  FILLER  PIC  X(10)  VALUE  '0270000320'.
    02  FILLER  PIC  X(10)  VALUE  '0290000470'.
    02  FILLER  PIC  X(10)  VALUE  '0300000290'.
    02  FILLER  PIC  X(10)  VALUE  '0330000960'.
01  UNIT-PRICE-TABLE-R  REDEFINES  UNIT-PRICE-TABLE.
    02  TABLE  OCCURS  20  TIMES.(将上面数据分配给各个表元素)
        03  PRODUCT CODE-T  PIC  9(4).
        03  UNIT-PRICE-T    PIC  9(6).
PROCEDURE      DIVISION.
PROGRAM-START.
     OPEN  INPUT    OLD-SALES-FILE.
           OUTPUT   NEW-SALES-FILE.
READING.
    READ  OLD-SALES-FILE.
          AT  END  CLOSE  OLD SALES-FILE、NEW-SALES-FILE
                   STOP  RUN.
    MOVE  YEAR  TO  YEAR-N.
    MOVE  MONTH  TO  MONTH-N.
    MOVE  DAYY  TO  DAYY-N.
    MOVE  PRODUCT-CODE  TO  PRODUCT-CODE-N.
    MOVE  CUSTOMER-CODE  TO  CUSTOMER-CODE-N.
    MOVE  QUANTITY  TO  QUANTITY-N.
    PERFORM  TABLE LOOKUP  VARYING  I  FROM  1  BY  1  UNTIL  I  >  20.
    GO  TO  READING.
TABLE-LOOKUP.
```

```
IF  PRODUCT-CODE=PRODUCT-CODE-T (I)
    MOVE  UNIT-PRICE-T(I)  TO  UNIT-PRICE-N
    COMPUTE  SALES-VALUE-N=QUANTITY-N  *  UNIT-PRICE-N
    WRITE  NEW-SALES-RECORD.
```

说明：内部文件 OLD-SALES-FILE 和磁盘数据文件 IN-FILE. DAT 相联系，NEW-SALES-FILE 和 P-FILE. DAT 相联系。这种联系是在环境部中描述的。在数据部的文件节中分别对两个文件的记录进行数据描述。在工作单元节中，定义“I”作为“表”的下标，以后每次给 I 赋予一个值，以便查单价表。用上述的方法定义了单价表 TABLE-R，并对它们赋予初值。

在过程部中先将读入的数据传送到磁盘文件记录区。见图9.8。在磁盘文件记录中还有

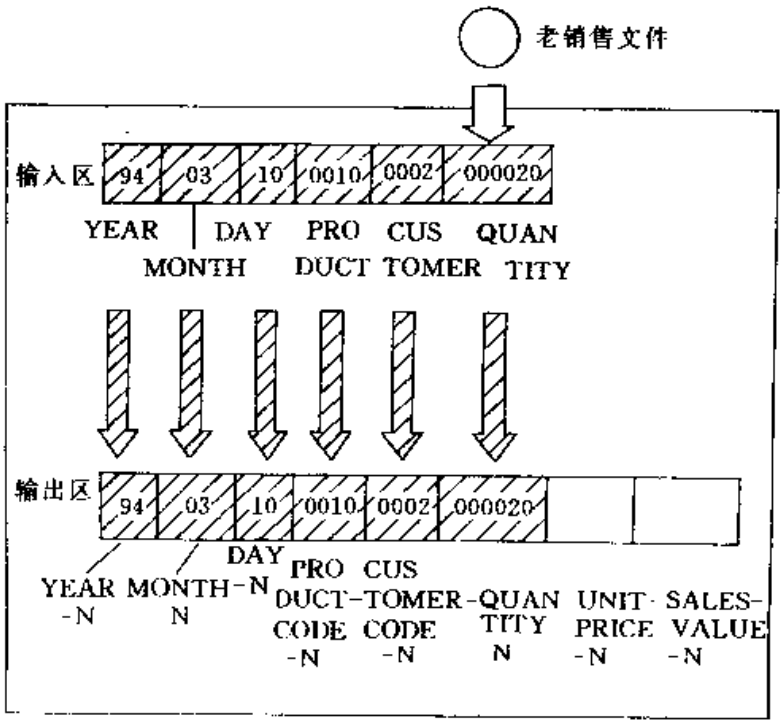



图 9.8

两项（UNIT-PRICE-N，SALES-VALUE-N）未被赋值。为了找出单价，需查单价表，即 TABLE 表。从表的第一个元素查起，将从记录读入的货号 PRODUCT-CODE 逐个与各表元素中的 PRODUCT-CODE-T 相比（I 从1逐次变化，每次加1）。如果查到某一个表中元素的货号 PRODUCT-CODE-T（I）与从记录上读入的货号 PRODUCT-CODE 相同，则将该货物的单价送到输出文件记录区中的 UNIT-PRICE-N，并计算出该货物的款项（数量×单价），送到输出文件记录区中的 SALES-VALUE-N 中去。见图9.9。然后写到磁盘文件上去。

假如找遍表中20个元素，都找不到所需货号，表示此货物不存在，不予处理。再读入下一个记录。查表和处理的流程见图9.10。每读入一个记录，都要重新查一次表，每次查表都从第一个表元素开始，因此必须每次重新使 I 的初值等于1。

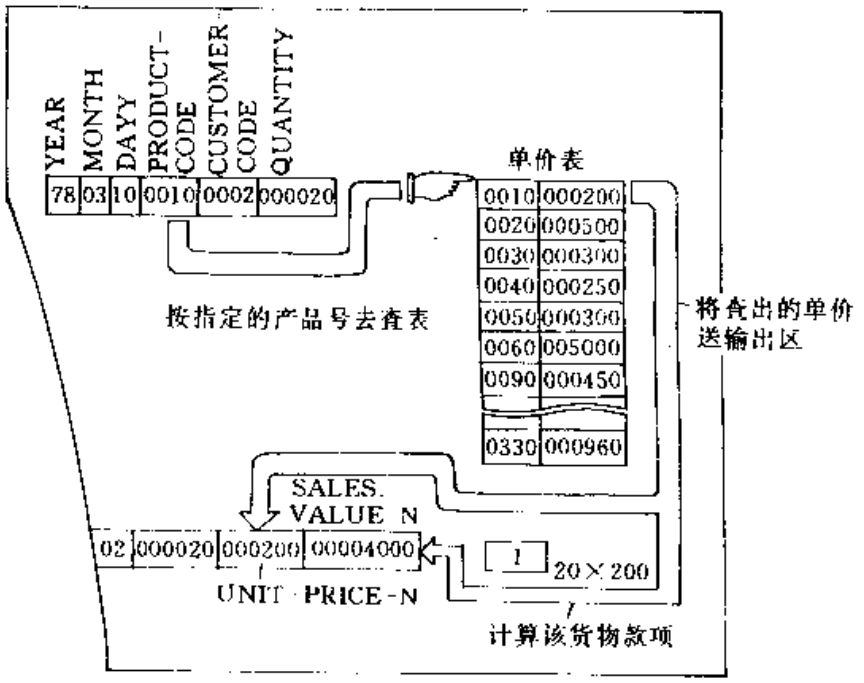



图 9.9

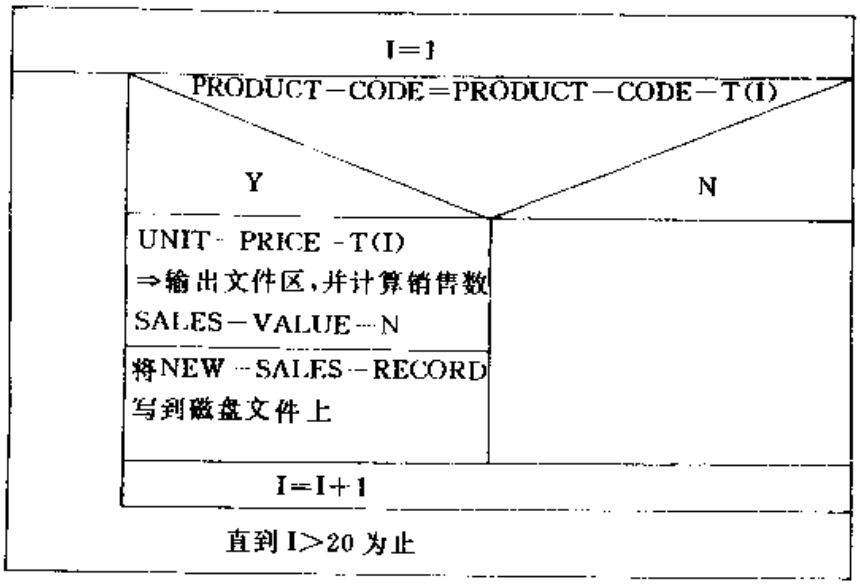



图 9.10

## §9.7 用位标法引用表元素

除了可以用指定表名和下标的方法来引用一个表元素外，COBOL 还提供另一种方法，即位标（Index。又译作足标，变址下标，索引下标，指标等）法。所谓“位标”是一种特殊类型的数据，专用于表元素的引用。

### 9.7.1 位标的概念

我们前面用的“下标”是代表元素在表中的序号。例如 A (3)，表示 A 表中第三个元素。而“位标”是代表一个表元素在表中的相对位置。在许多计算机中，是按相对地址来计算的。

以表的第一个元素的第一个字节作为相对地址0。位标就是所指定的元素的第一字节的相对地址，即它对表的起始位置的位移量。

假如有一个表，包含10个元素，每个元素占有三个字节。则第一个元素从相对地址0开始。第二个元素从相对地址3开始，……第五个元素从相对地址12开始。相对地址的计算公式是：（表元素的序号－1）×表元素长度。本例中，如果表元素的序号为5，表元素长度（占字节数）为3，则相对地址＝（5－1）×3＝12。见图9.11。它在内存中的绝对地址等于相对

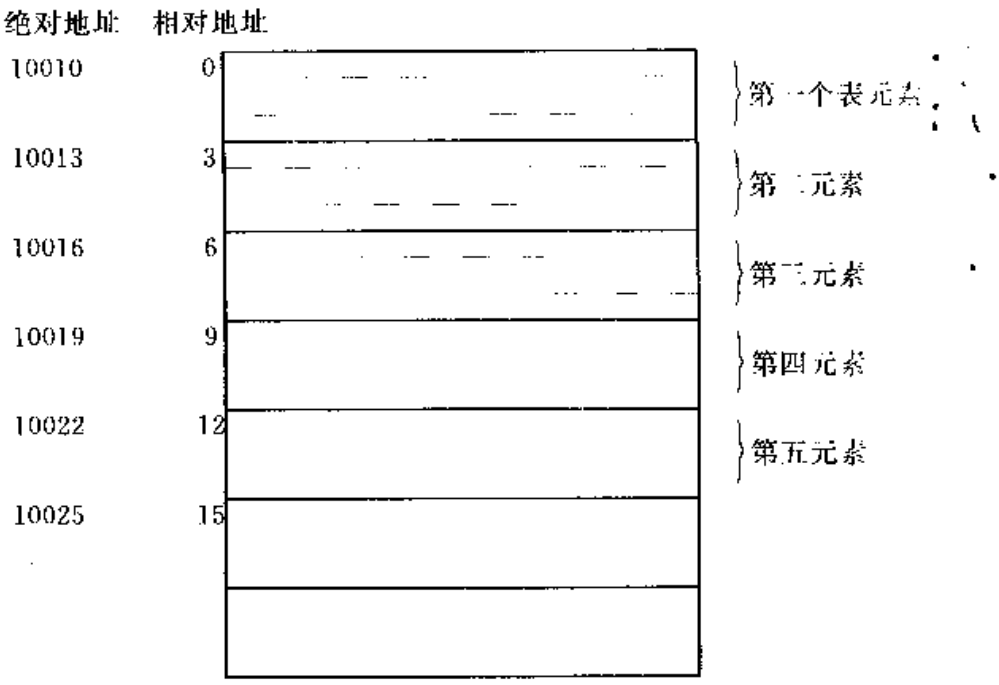



图 9.11

地址加上表的开头地址。假如表的起始地址为10010，则上述表中第五个元素的绝对地址就是10010＋12＝10022。

因此，位标的值是表示表元素在该表中的相对位置的（以字节数表示）。称它为“位标”要比“足标”、“指标”、“索引下标”等更能确切地表示它的含意。

如果表元素的长度变成4个字节，则第五个表元素的相对地址就变成（5－1）×4＝16了。由此可知，位标的值不仅与表元素的序号有关，还和表元素本身长度有关。

用下标法和位标法都可以引用一个表元素，它们之间有什么区别呢?以前介绍的下标法，是直接用序号（如A（3））来引用表元素，编译系统根据这个序号计算出需引用的表元素的地址，有了地址才能把表元素的值调出来。如果多次写：MOVE A（3）TO B，则每引用一次A（3）都要由系统进行一次地址的计算。而用位标法引用表元素（例如写A（I），I是已被指定为位标的数据项名），先给I送入一个值（这个值就是相对地址），则在执行MOVE A（I）TO B时，不需每次都进行相对地址的计算了，直接按地址找到该表元素。

### 9.7.2 位标名的指定方法

用下标法引用元素可以写成A（3）形式，也可以写成A（S）形式。其中S是一个作为下标的数据名。用位标法与之相仿，如A（I），其中I是位标名。那么怎样区别哪一个数据名是下标，哪一个数据名是位标呢?对作为位标的数据名要专门加以指定。在数据部中定义一个表时所用的OCCURS子句中要加上“INDEXED BY 位标名”短语来指定。如：

```
01  A.
    03  T  OCCURS  5  PIC  X(2)  INDEXED  BY  I.
```

这表示表的名字是T，它有五个元素，每个表元素都是字符型数据占为二个字节。“INDEXED BY I”表示指定I作位标名，以后引用T表元素时，可以用I作位标。只要出现T(I)，系统便知道按位标来处理I。即把I的值作为相对地址来处理。

需要特别指出的是，这个位标（例如上面的I）不需要在任何其它地方对它进行数据描述，例如：

```
77  I  PIC  9(3).
```

这反而是画蛇添足了。只需要在OCCURS BY 短语中指定一个位标的名字就足够了。它是一种特殊类型的数据，专用于表元素的访问。系统会专门为它分配内存空间和指定存放数据的形式，用户可不必过问。

说明：（一）凡以后需要用位标来引用表元素的，都需要在建立（定义）表时用“INDEXED BY”短语来指定位标的名字。如果不经指定，不作为位标处理。例如上例中如不用“INDEXED BY I”，则以后出现T(I)时，不将I作位标对待，而将它作为一般的数据项处理（认为用一般的数据项I来作下标）。

（二）由于位标是专用于引用表元素的特殊的数据项，它不能用来进行算术运算，如ADD 2 TO I，COMPUTE I=I+1等都是错误的。也不能由输入语句（READ）来改变它的值（如从磁盘文件读入一个值给I是不行的）。也不能将它的值直接打印或显示出来。它不是记录的一部分，不能在文件节定义，也不必在工作单元节中定义。

（三）一维表或多维表的每一维按需要可以指定若干个位标名，引用时这些位标名只能在该维内使用。即按维定义，按维使用。这就是为什么位标名要在OCCURS子句中定义的原因。例如：

```
01  A.
    02  B  OCCURS  5  INDEXED  BY  B1,B2.
        04  C  OCCURS  4  INDEXED  BY  C1,C2.
            05  D  PIC  9(2)  OCCURS  8  INDEXED  BY  D1,D2,D3.
```

若要引用表元素，在引用B表元素时可以用B1，或B2作位标，在引用C表元素时相应的维可以用C1或C2作位标，在引用D表元素时相应的维可以用D1，D2或D3作位标。例如：B (B1)，B (B2)，C (B1，C1)，C (B2，C1)，D (B2，C2，D3)，D (B1，C2，D2) 等都是正确的。即第一维可用B1，B2，第二维可用C1，C2，第三维可用D1，D2，D3。而下面的用法是不合法的：B (C1)，C (C1，D2)，D (D1，D2，D3)。

（四）有时需要把位标的值转存到另一个数据项中，但由于位标是特殊类型的数据项，因此，需要另外定义一种特殊的数据项叫位标数据项，用来专门存储位标的值。位标数据项在数据部中定义。描述位标数据项用USAGE子句。例如：

```
77  K  USAGE  IS  INDEX.
```

表示数据名K是用来作位标数据项的。不必再加PIC子句（如PIC 9(4)等），由系统特殊分配内存空间和指定存储方法。

为了不使读者搞混淆，我们复习一下对数据项用USAGE子句的情况：

```
77  A  PIC  9(4)  USAGE  IS  DISPLAY.(表示A是用来作显示的，一个字符占一字节)。
77  B  PIC  9(8)  USAGE  IS  COMP.(表示B是按二进制数方式存放数据的)
```

77 C USAGE IS INDEX.(表示C是用来存放位标值的特殊数据项)

上述各描述体中“USAGE IS”可以省写。如：

77 C INDEX.

有的计算机系统对这种位标数据项一律分配一个机器字（四个字节），用二进制表示。位标数据项与位标一样，不能参加算术运算。

### 9.7.3 SET（设置）语句

由于位标不是普通数据项，因此不能用向一般数据项赋值的方法向它赋值，如果I已被指定为位标。则：

MOVE 10 TO I.

COMPUTE I=A+B.

等是不合法的。

专门有一SET语句来给位标赋值，其作用是将一表元素的相对地址放到指定的位标去。例如：

SET I TO 10.

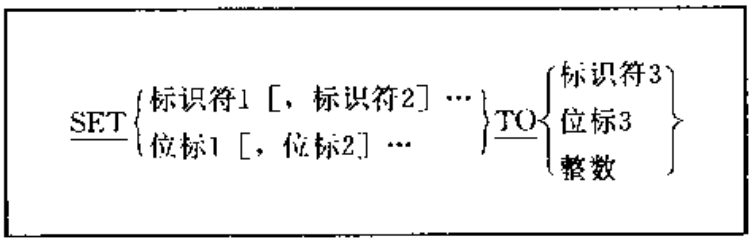
表示“将位标I置位到第10个元素的第一个字节的相对地址位置上”。初学时容易有两点易搞错：①误理解为“将I送到10去”。②误以为直接将数10简单地送到I中去。

实际上，是将相对地址送到I中，10只是表示表中第十个元素，要把它化成相对地址。编译系统先查出I是作为哪个表的位标，查出该表元素的长度，再进行计算，如：

```
01 A.
  02 B OCCURS 10 PIC X(10) INDEXED BY I,J.
  02 C OCCURS 10 PIC X(2)  INDEXED BY K,L.
```

上面的I是用于B表的位标，而每个B元素占10个字节。计算B表中第10个元素对B表的起始位置的相对位置=(10-1)×10=90，将90⇒I。

（一）SET语句的一般格式（之一）




分别举例说明之（以上述定义的表为例）。

(1) SET I TO J

I和J都是位标名，将I“置位”于J，或认为“将J中的值送到I中”。如果J的值为90，则I也是90。

(2) SET J TO 5

将出现号5化成用J作位标的表（表B）当中的相对地址（(5-1)×10=40），然后将40送给J。

(3) SET I TO D

如果D为一般数据项，I为位标名。若D的值为10，则按上述原则将10转算成B表中相

对地址 90 送 I。

(4) SET K TO I

假如 K 已被定义为位标数据项，即：

77 K INDEX.

SET 语句要求将位标 I 的值存到位标数据项 K 中，作简单传送。如 I 值为 90，则传送后 K 值也是 90。

(5) SET J TO K

将位标数据项 K 的值简单地送到位标名 J 中，如果 K 的值为 90，则 J 的值也是 90。

(6) SET D TO I

将位标 I 中的值（相对地址）化成表元素的顺序号，再传送给一般数据项 D。

今 B 表元素长度为 10，则顺序号为：90÷10＋1－10，再将 10→D。

(7) SET L TO I

I 和 L 都是位标名，但它们分别是在 B 表和 C 表的描述体中定义的。而 B 表元素含 10 个字节，C 表元素含两个字节，如果 I 的值为 90，不能简单地将 90⇒L。而是先将 90 化成序号 (90÷10＋1＝10)，即它指的是 B 表第 10 个表元素。然后计算出用 L 作位标的 C 表中第 10 个元素的相对地址：(10－1) ×2＝18，最后将 18⇒L。即由 B 表中的相对地址转换成 C 表中相同序号的表元素的相对地址。

它常用于这种情况：当我们对第一张表 B 执行了查找并求得了所需的元素后，可能希望引用第二张表 C 的对应元素。譬如，已找出的是 B 表的第三个元素，则通过 SET L TO I，就可以通过 L 引用 C 表的第三个元素。

注意：不能用 SET 语句将一整数送到位标数据项或将一般数值数据项或整数送到数值数据项中，例如：

SET K TO 10. (K 为位标数据项)

SET D TO T. (D，T 为一般数值型数据项)

SET D TO 3. (D 为一般数值型数据项)

是不合法的。

为查阅方便，将 SET 传送规则列表如下（设 SET A TO B)：

| 接收项 A ＼ 发送项 B | 整数 | 数值初等项 | 位标名 | 位标数据项 |
|---|---|---|---|---|
| 数值初等项 | × | × | 将位标代表的序号传送 | × |
| 位标名 | 将整数(序号)化成相对地址传送 | 同左 | (1) 如果两个位标名指向同一表，则简单传送<br>(2) 如不指向同一表，则转换成另一表的相对地址再传送，即将发送项代表的表元素顺序号减 1，乘以接收项相关的表元素的长度，再传送。 | 简单传送 |
| 位标数据项 | × | × | 简单传送 | 简单传送 |

注：表中“×”表示不合法。

在传送中的转换是系统自动进行的，用户可不必过问。对用户来说，可以简单地认为位标法和下标法一样地把序号传送给位标，然后按此序号找出表元素。从实际效果来看，二者的结果是一样的。例如：

① MOVE 3 TO D （D为数值数据项）
DISPLAY T (D).

② SET I TO 3 （I为位标名）
DISPLAY T (I).

二者都是显示出T表中第三个元素的值。

我们只是为了说明位标的概念才对其传送过程作了比较详细的解释。有人可能问，既然如此，为什么有了下标，还要用位标呢？原来设想利用位标查找表元素效率可能高些，但实际上效果并不显著，而使用起来规则限制很多，很不方便。用位标法还不利于调试，因为不能将位标值用DISPLAY语句显示出来。如果要知道位标I所代表的序号。要先用SET语句转换一次：

SET T TO 1.
DISPLAY T.

因此，除了在下节（§9.8）将要介绍的SEARCH（检索）语句中使用位标法以外，一般情况下使用位标来引用表元素是不多的。

（二）SET语句格式（之二）

```
SET 位标1 [，位标2]... { UP   BY } { 标识符 }
                       { DOWN BY } { 整  数 }
```

用它来给位标加减一个量。如：

SET I UP BY 2.

把位标I代表的序号加2。如果I原来是代表序号为10的相对地址，则变成代表序号为12的表元素的相对地址。对用户来说，可不必细算，简单地理解为原来I指向第10个表元素，今改为指向第12个表元素。

同样，SET I DOWN BY 1. 表示将I代表的序号减1。

SET I，J DOWN BY 2. 表示I，J所代表的序号都减2.

总之，与MOVE不同，SET是专门用来进行位标值的传送的。有的读者容易把位标名和位标数据项两者混淆。下面简单对比一下：

① 02 A OCCURS 10 PIC X(5) INDEXED BY A1，A2，A3。

A1，A2，A3是位标名，可用它来引用表元素。

② 77 D USAGE INDEX.

D为位标数据项，它可以用来存放位标的值，但不能直接用它作为位标来引用表元素。如A (D)的用法是不对的。因在定义A表时只指定了A1，A2或A3之一可以在引用该表时作位标。如果要取用D值，则应：

SET A1 TO D.
MOVE A (A1) TO OUT-REC.

先将 D 值送给 A1，再用 A1 作位标引用 A 表元素。

### 9.7.4 使用位标引用表元素的方法

（一）使用位标有两种方法：一是在表名后的括弧内直接写上位标的名字，如已见到过的 A（A1）。二是在括弧中使位标名加或减一个整数，如 A（A1+3），如果 A1 当前值指向的是第二个表元素，则加 3 变成指向第五个表元素。读者可能记得用下标法引用表元素时，括弧内不能是表达式。如 A（3+4）是不对的。

（二）位标所代表的“出现号”不能超过表的相应维的长度。如果 A 表有 10 个元素，则位标 A1，A2，A3 所代表的元素序号都只能在 1～10 之间。如用上面（一）中的第二种方式引用表元素，则应注意，加（减）一个整数后，其代表的元素序号不应小于 1 和大于表的元素个数。如果 A1 的当前值是指向第二个元素，则 A（A1+9）代表第 11 个元素，超过 10，故不合法。

（三）一个表，不论是否指定位标，都可以使用下标。

如：02 A OCCURS 5 INDEXED BY A1，A2，A3.

03 B OCCURS 10 PIC X(8) INDEXED BY B1，B2，B3.

既可以用 A（A1），A（A2），A（A3），也可以不用位标，而用下标引用表元素。如 A（1），A（3），A（D）等（D 是数值数据项，值在 1～5 之间）。

但位标不能与下标混用，如 B（A1，B1），B（D，T）合法，但 B（A1，D）或 B（T，B2）不合法。因 D，T 是作下标用的一般数值数据项，而 A1，B2 是位标，不能在引用一个表元素时同时用位标和下标，因为它们是用不同的方法处理的。

但允许位标与常数联用，如 B（A1，3）或 B（4，B3）合法。

（四）位标可以用在 IF 语句中参加比较，如 IF I ＞3…。如果位标与一个非位标数据项比较，系统会自动将位标值转换成它所代表的元素序号然后比较。用户可以把它看作和下标一样比较即可。即如果 I 指向第五个元素，我们简单地把 I 看作是序号 5 即可，将 5 与 3 比较。这样好理解些。

## §9.8 表的检索

要找一个表元素，最简单的方法就是指出表名和下标（或位标），直接找到所需的表元素，这叫直接法。但是有时我们只给出所需表元素的某些特点，而不知道它是第几个元素。例如，一个职工工资的表，包含 100 个职工的姓名、性别、工龄、工资等数据，我们希望把工龄在 10 年以上而工资低于 150 元的职工名单打印出来，以便调整工资时参考，但我们事先不知道职工表中第几个元素是所需要的。又如，有一户口管理的表，包含某个地段的全部人口的姓名、性别、工作单位、职业等。今因某种需要，希望很快查出本地段中名为“张生”的，男性，在人民钢厂工作的人，这样就无法用直接法直接找出所需的表元素了。

当然可以用本章中前面举过的查单价表的例 9.1 的方法，编一段程序，找一个表元素出来，用 IF 语句判断所要求的条件是否满足，一个一个地找下去，直到找到为止。但这样做，程序比较复杂，处理效率也不高。

COBOL 提供了 SEARCH（检索）语句专门解决这一类问题。

### 9.8.1 用于顺序检索的 SEARCH 语句

假如有一个表，其表元素的排列是没有规律的。为了要找出满足给定条件的表元素，只能一个一个地顺序检查表中各元素，直到找到为止。这叫顺序检索法。

举一个简单的例子来说明 SEARCH 语句的功能。

**【例 9.2】** 假如某一个单位，有 50 个职工，已建立了以下的表：

```
01  WORKER-REC.
  02  WORKER-TABLE  OCCURS  50  TIMES  INDEXED  BY  N.
    04  NUMB  PIC  9(4).
    04  NAME  PIC  X(20).
    04  AGE  PIC  9(3).
    04  PAY  PIC  9(3).
```

要求找出其中姓名为“张生”的职工的工资，我们可以在过程部中写以下语句：

```
SET  N  TO  1.
SEARCH  WORKER-TABLE      (表名)
      AT  END  DISPLAY  'CANNOT  FIND  NAME'
  WHEN  NAME (N) ='ZHANG  SHENG'
        DISPLAY  NAME (N),  PAY (N).
  ⋮
```

为了从表 WORKER-TABLE 的第一个元素查起，先用 SET N TO 1（如果想从第5个元素查起，则 SET N TO 5）。

SEARCH 语句是检索语句，它顺序检索 WORKER-TABLE 表中的各元素，直到“WHEN”后面指出的条件满足为止。今指定的条件是 NAME (N) = 'ZHANG SHENG' 即将 'ZHANG SHENG' 逐个地和 NAME (1)，NAME (2) …比较，直到找到表中某一个元素 NAME (N) 等于“ZHANG SHENG”为止。此时，显示出他的名字和工资。如果找完50个表元素，都找不到这样的名字，则执行 AT END 后面的语句，显示“找不到此名字”，然后跳过 WHEN 子句继续执行下一个句子。

应当指出：SEARCH 后必须跟一被检索的表名。而且用 SEARCH 语句必须用位标法来引用表元素。因此，在 WORKER-TABLE 表的描述体中必须带有“INDEXED BY”子句。本例中应指定 N 为位标名。在 WHEN 后面的条件中必须出现用位标法引用的表元素。如 NAME (N)，N 正是前面指定的位标名，如果象下面这样写就不对了：

```
MOVE  1  TO  D.
SEARCH  WORKER-TABLE
    AT  END  DISPLAY 'CANNOT  FIND  NAME'
    WHEN  NAME (D) ='ZHANG  SHENG'
      DISPLAY  NAME (D), PAY (D).
```

因为 D 不是位标名，不能用于 SEARCH 语句的条件中。

SEARCH 语句是这样执行的：从指定的表元素开始，检查是否满足 WHEN 后面指定的条件。如不满足，就使 N 增值，自动执行一个 SET N UP BY 1。使 N 指向下一个元素的地址。如果查到某一个元素满足指定的条件时，查表工作即停止，执行 WHEN 子句中条件

后面的语句。此时 N 保持当前的值不变。有的读者可能会问：不是规定位标值不能用 DISPLAY 显示出来吗?怎么现在可以用 DISPLAY NAME (N)，PAY (N) 呢?我们说位标是特殊类型的数据项，不能用 DISPLAY 语句来直接显示位标的值，如 DISPLAY N 是不行的。但 NAME (N) 已是一个表元素了，它是用位标法引用的表元素，是一个字符型的数据项，当然，可以用 DISPLAY 显示。

如果省略不写 AT END 子句，则在查完整个表还找不到满足指定条件的表元素时，就执行 SEARCH 语句的下一个句子，相当于写了 "AT END NEXT SENTENCE"。

如果在查表开始时，位标的值所代表的元素的序号已大于表元素的个数（如本例 SET N TO 60），则查表不会进行。

有时希望把满足几个条件之一的表元素找出来。

**【例9.3】** 工厂中检查车间工作，要求把产量低于500（定额）的或产品质量等级为 "D"（不及格）的打印出来。

先在数据部中定义 "表"

```
01  PRODUCT-TABLE.              (产品表)
  02  SHOP-PRODUCTION (车间生产情况) OCCURS 10 INDEXED BY I.
    04  NUM  PIC  9(6).         (车间号)
    04  QTY  PIC  9(4).         (数量)
    04  GRADE  PIC  X.          (等级)
```

在过程部中用以下语句：

```
    ⋮
   SET  I  TO  1.
SEARCHING.        (段名)
      SEARCH  SHOP-PRODUCTION      (表名)
         AT  END  GO  TO  FINISH
         WHEN  QTY (I) <500  DISPLAY  NUM (I), QTY (I)
         WHEN  GRADE (I) ='D'  DISPLAY  NUM (I),
            GRADE (I).
      SET  I  UP  BY  1.
      GO  TO  SEARCHING.
```

SEARCH 语句检查每一车间的产品数量是否<500，或者产品质量是否为 "D" 等，满足第一个条件的则执行第一个 DISPLAY 语句，如满足第二个条件，则执行第二个 DISPLAY 语句。其流程图见图9.12。

两个 WHEN 子句，不是表示 "两个条件同时成立"（即 "与" 条件），而是指只要满足两个 WHEN 子句中条件之一，就认为 "找到了"，此时执行 WHEN 后面指定的语句 (DISPLAY)，然后停止查表。请注意，如果10个车间中第三车间和第五车间的产量<500，而第八车间质量为 D 等，则执行一个 SEARCH 语句只能找出第一个满足两个条件之一的车间（第三车间），然后停止查表。后面两个车间找不出来。因此，要从第四车间接着继续往下查找。所以在 SEARCH 语句下面加一个：SET I UP BY 1。使 I 指向下一个车间，继续找，直到找完十个车间后执行 AT END 子句。

SEARCH 语句的一般格式（之一）：

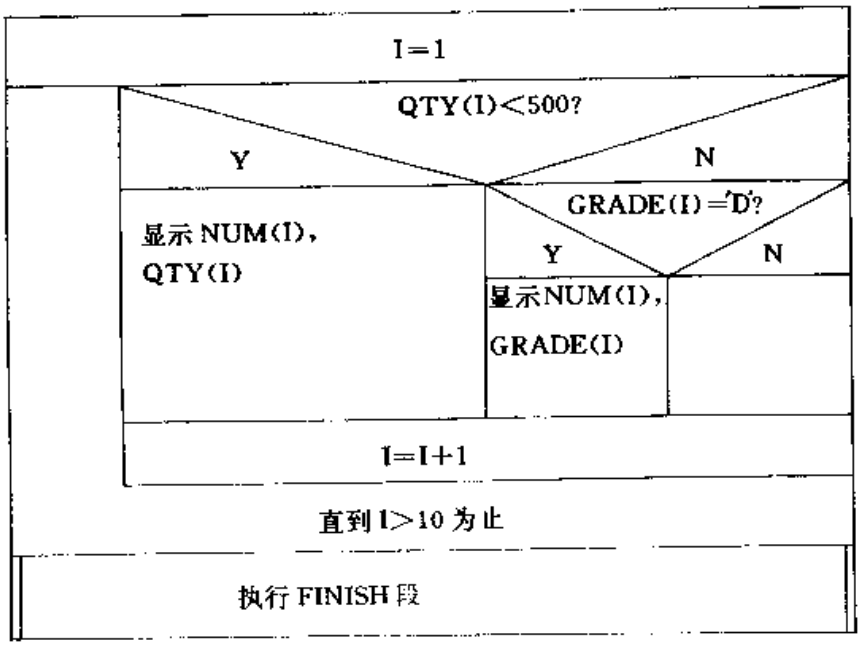



图 9.12

```
SEARCH 表名 [VARYING {位标名1 / 标识符2}] [AT END 强制语句1]
        WHEN 条件1 {强制语句2 / NEXT SENTENCE}
        [WHEN 条件2 {强制语句3 / NEXT SENTENCE}]...
```

上面格式中有一个 VARYING 可选项。有以下几种情况：

(1) 如果不选用此 VARYING 可选项（如例9.2和例9.3），则表示检索语句是按定义这个表时所用的“INDEXED BY 位标1，位标2……”中的位标1来查表的（如例9.3中的 I）。假如例9.3中指定了两个位标 INDEXED BY I，J。由于未用 VARYING 项，按规定隐含用第一个位标名 I 来检索，每次自动使 I 增值。条件中用的是 QTY (I)，GRADE (I) 是可以的。

(2) 如果用 VARYING 项。譬如 VARYING J，而这个 J 是属于 SHOP-PRODUCT 表的。则查表时应按位标 J 来进行。即应该使用 QTY (J)，GRADE (J)。在执行 SEARCH 语句过程中，每次自动使 J 增值，而 I 值不变。

(3)如果用的是另外一个表的位标名，如 VARYING K。而 K 并不是 SHOP-PRO DUCTION（车间生产情况）表所指定的位标，而是属于另一个 SHOP-PAY（车间工资）表的一个位标，如：

```
01  PAY-TABLE.
  02  SHOP-PAY  OCCURS  10  INDEXED  K, L.
    03  NUMB  PIC  9(3).
    03  PAY  PIC  9(5).
```

则在用 SEARCH 查表时，仍用被查的表（SHOP-PRODUCTION）所指定的第一个位标（即 I）来进行，这和没用 VARYING 可选项相同，但当 I 每次增值时，VARYING 后面的

那个位标K也随之增值，即指向它相关的那个表的下一个表元素。譬如，找出第三车间未完成产量任务，此时I指向SHOP-PRODUCT表中第三个表元素，K指向SHOP-PAY表的第三个元素。这有什么用呢?我们可以在“车间生产情况”表中查出哪一车间未完成任务时，可以马上从车间工资支付表中找到相应的表元素，扣除该车间的总工资20%等等。如：

```
SEARCH  SHOP-PRODUCTION
    VARYING  K  AT  END  GO  TO  FINISH
    WHEN  QTY (I) < 500
      MULTIPLY  0.8  BY  PAY (K).
```

在这种情况下巧妙地用VARYING可选项，可以使程序简单。

(4) VARYING后面跟的不是位标名，而是普通的数值数据项(例如VARYING D)，则仍按被查的表所指定的第一个位标(I)来查表，当I增值时，这个数据项D也每次随之加“1”。它的作用是把位标I所指向的元素的序号保存下来，以便需要时调用。

执行上页SEARC语句的流程见图9.13。图中“使位标值增1”的含义是：位标值增加一个表元素的长度（字节数）。

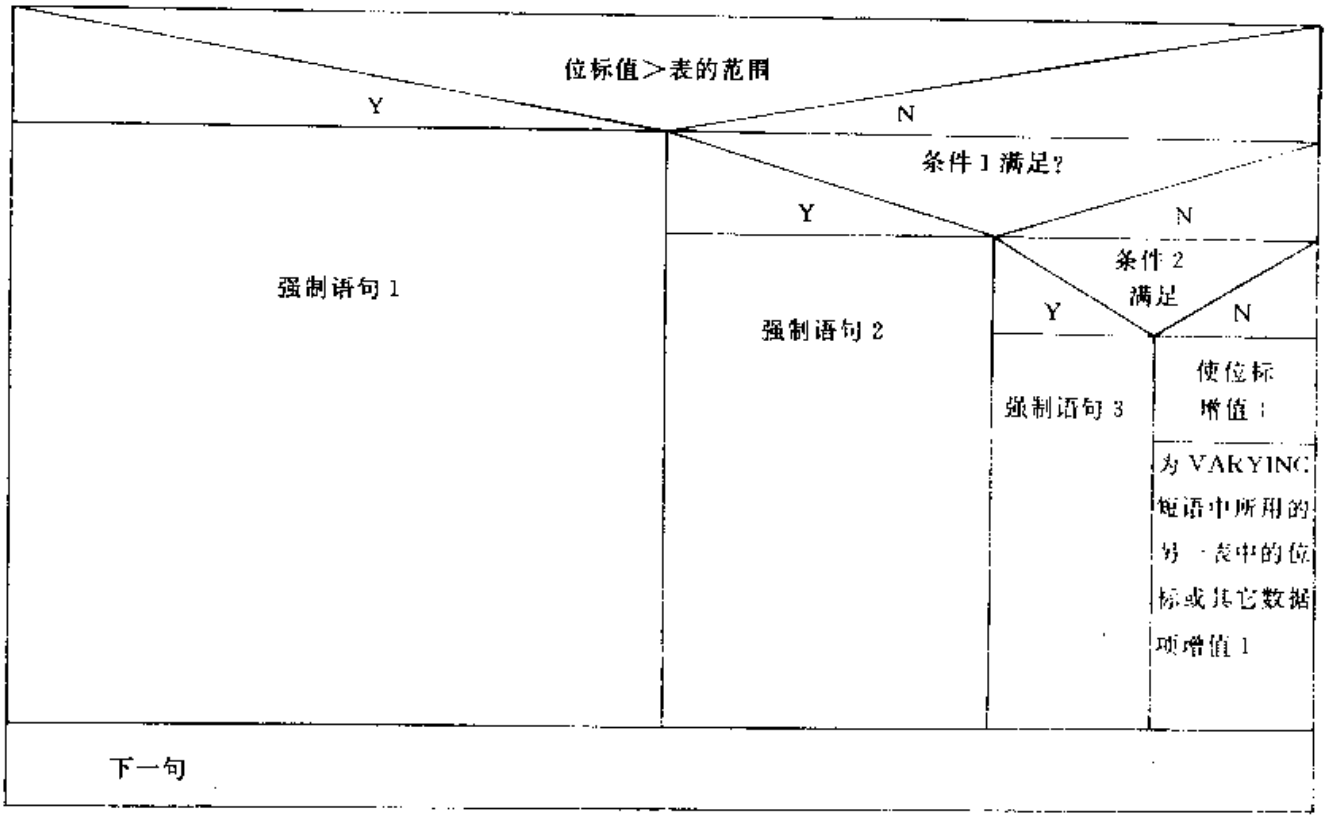



图 9.13

## 9.8.2 用于有序表的SEARCH语句

上面格式的SEARCH语句是顺序检索表元素，如果被检索的表包含表元素很多，这样的方法效率是比较低的，假若有一个表，它的表元素是已按某一规律排列好的，则可以用另一种格式的高效率的SEARCH语句。

（一）折半检索法的概念

譬如，有一个图书目录的表，包括书名、编号、分类、存放书架的编号。假设表中各元

素已按书名的字母顺序排列好。今想查一本名为“COMPUTER DESIGN”的书，是属于哪一类，放在哪一个书架上，以便借阅。我们不必从第一个表元素（即A字母开头的书）查起。许多计算机系统提供了“二分法”（或称折半法）检索。即先找中间的一个表元素。如果当中的一本是“NEW CHINA”，系统经过比较，发现“C”在“N”之前，由于书名是按字母顺序排列的，因此对“N”以后字母开头的不必检索。见图9.14的①部分。再对第一本书和“NEW CHINA”之间的若干本书取当中的一本，如果它是“GREAT WALL”，由于C在G之前，则又把“GREAT WALL”后面的舍弃，见图9.14的②部分。再取第一本和“GREAT WALL”之间当中的一本，假设为“BASE BALL”，由于B<C，显然“COMPUTER DESIGN”不会在“BASE BALL”之前，故又将图中③部分舍弃，……如此继续，每次取中间的一个表元素，直到找到为止。显然这种折半检索法比顺序检索法效率高多了。假如我们有100本书，想找的是第75本，则先找50，舍弃1～49部分，再找50～100之间的当中一个，即找出75。一共两次就找到了。1～100之间任何一本书，最多找的次数为：$\log_2 N$。N为表元素个数，今N=100，$\log_2 N=\log_2 100-6.65$，即最多找7次。读者可自己试一试，看有没有超过7次还未找到的。

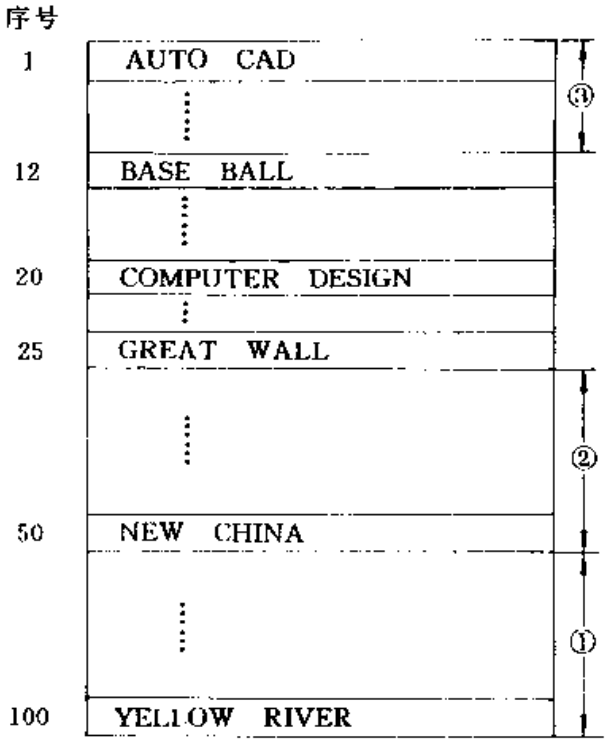



图 9.14

（二）ASCENDING KEY和DESCENDING KEY

用折半法检索的表，表元素必须是根据其中某一个（或几个）项的值严格递增（或递减）的次序排列好的。否则如何判断该舍弃哪部分表元素呢？查找就是以这个递增（减）的数据项作依据的（象上面例中就是以“书名”来查找的）。

在建立表的时候要声明此表是已按什么规律排列的。如：

```
01  BOOK
  02  BOOK-TABLE  OCCURS  100  TIMES        (表包括100元素)
        ASENDING  KEY  IS  NAME              (按书名升序)
```

```
        INDEXED  BY  N.                    (位标为 N)
    05  NAME  PIC  X(20).                  (书名)
    05  CLASS  PIC  X(2)                   (分类)
    05  COD    PIC  X(4)                   (编号)
    05  NUM    PIC  9(3)                   (书架号)
```

说明了 BOOK-TABLE 表的100个元素是以 NAME（书名）的升序排列的。“ASENDING”是“递升”的意思。如果是递减（降序），则用“DESCENDING”，显然，NAME 必须是表元素中的一个数据项。有的初学者以为写了 ASENDING KEY 以后，系统就会自动将这100个表元素按书名排列了。这是不对的。建立表时是没有自动排序的功能的；在建立表时声明“ASENDING KEY IS NAME”，只是通知计算机：“BOOK-TABLE 表中各元素已按数据项 NAME 的升序排列好了”。程序编制者必须保证表中元素已按此规律排列好，否则就会出错。也就是说，用折半检索法的对象是已排好序的表。

说明升（降）序的一般格式为：

```
{ASCENDING  }
{DESCENDING } KEY  IS  数据名1 [，数据名2] …

    [ { ASCENDING  }                                    ]…
    [ { DESCENDING } KEY IS  数据名3 [，数据名4] …      ]…
```

即可以有几个排序项，如：

```
02  BOOK-TABLE  OCCURS  100
        ASCENDING  KEY  IS  NAME
        DESCENDING  KEY  IS  COD
        INDEXED  BY  N.
```

说明这个表中各元素以书名为“主键”升序排列，如书名相同，则再按编号的降序排列。

（三）SEARCH 语句格式（之二）

折半检索是用第二种格式的 SEARCH 语句来实现的。在动词 SEARCH 后面加一个“ALL”。如：

```
SEARCH  ALL  BOOK-TABLE
     AT  END  DISPLAY 'CANNOT  FIND  NAME'
     WHEN  NAME (N)  =  'COMPUTER  DESIGN'
           DISPLAY  NAME  (N), NUM  (N).
```

此语句的功能是：用折半检索法检索表 BOOK-TABLE，直到找到某一表元素中的书名（即 NAME（N）的内容）等于“COMPUTER DESIGN”为止，找到后显示该书名和书架号。如找完该表还找不到满足此条件的表元素，执行 AT END 子句中的语句，显示“找不到此书名”的信息。

SEARCH 语句的一般格式（之二）的简单形式为：

```
SEARCH  ALL  表名 [; AT  END  强制语句1]
             WHEN  条件  {强制语句2      }
                         {NEXT  SENTENCE}
```

使用此格式的 SEARCH 语句的说明：

(1)“条件”中只允许用关系运算符“EQUAL TO”或“=”(等于)，不能写成 NAME (N) > “COMPUTER DESIGN”或用“<”等。

(2) 作为条件比较的一方（等号的任一侧）必须用 OCCURS 子句中指定的作为 ASCENDING KEY 或 DESCENDING KEY 项的数据项名，并且必须在其后的括弧中使用由 INDEXED BY 子句指定的位标名作“出现号”。例如，上例已指定按 NAME 升序(ASCENDING KEY) 排列，位标名定为 N，则条件中用 NAME (N) = 'COMPUTER DESIGN'，是正确的。如果不用 NAME (N) 来作比较，而用 CLASS (N) = 'AB'，这就错了。因为表元素是按 NAME 升序排列的，检索只能以 NAME 的值作比较才能判断应该舍弃不查哪一部分表元素。不用 NAME 作比较，就无法进行折半检索。也不能用 NAME (I) = 'COMPUTER DESIGN' 因为 I 不是位标名。

不能在等于号两侧都出现 KEY 项。例如，NAME 和 COD 分别指定作为 ASCENDING KEY 和 DESCENDING KEY 项，条件中不能用 WHEN NAME (N) =COD (N) 等形式。

(3) 由于是折半检索，系统会自动首先找出中间位置上的元素，不必用 SET 语句定位。

(4) ANSI COBOL 1974还允许用逻辑运算符“AND”将两个条件连接起来，表示“与”。即“两个条件同时满足”。但要求组成复合条件的第一个条件中使用 KEY 子句中第一个数据项名，第二个条件中使用 KEY 子句中第二个数据项名，看下例就清楚了。假如表定义时这样写：

```
02  BOOK-TABLE  OCCURS  100
        ASCENDING  KEY  IS  NAME
        DESCENDING  KEY  IS  COD
        INDEXED  BY  N.
```

则写 SEARCH 语句时可以这样：

```
SEARCH  ALL  BOOK-TABLE
    WHEN  NAME (N) ='COMPUTER  DESIGN'
      AND  COD (N)  ='0120'
      DISPLAY  NAME  (N), CODE (N), NUM  (N).
```

此处第一个条件用的 NAME 是上面 KEY 子句的第一个数据项名，第二个条件用到的 COD 是 KEY 子句中第二个数据项名，次序不能颠倒。这是由“表”元素的排列规律决定的。读者不难分析出其理由。它的检索是先查出书名为“COMPUTER DESIGN”的书（有时有相同的书名），再查第二个条件（编号为0120）。显然，这样检查的顺序应当和建立表时表元素排列所依据的“键”项的顺序一致。

如果一个 KEY 子句中指定有两个数据项，而 WHEN 子句中条件只用到一个，是可以的。

(5) SEARCH 语句一般格式（之二）的完整形式为：

```
SEARCH  ALL  表名  [AT  END  强制语句1]
    WHEN { 数据名1 { IS EQUAL TO } { 标识符3 }
                   { IS=          } { 常量1   }
                                    { 算术表达式1 }
         { 条件名1 }
       [ AND { 数据名2 { IS EQUAL TO } { 标识符4 }
                       { IS=          } { 常量4   }
                                        { 算术表达式2 } } ... ]
             { 条件名2 }
         { 强制语句2 }
         { NEXT SENTENCE }
```

最后应当说明：不是所有的COBOL编译系统都用折半检索法来实现SEARCH ALL格式的语句。本书以使用较多的折半法来说明，使读者有一个具体的概念，大致知道计算机系统是怎样实现这种语句的，而不致去死记各种规定。其实，对于用户来说，可以不管计算机用什么方法实现的，只要按语法要求正确使用即可。

### 9.8.3 程序举例

**【例9.4】** 在磁盘文件TABLEFILE中放了本年度的50个学校的按年龄分别统计的学生身体情况的数据，其格式如图9.15所示：

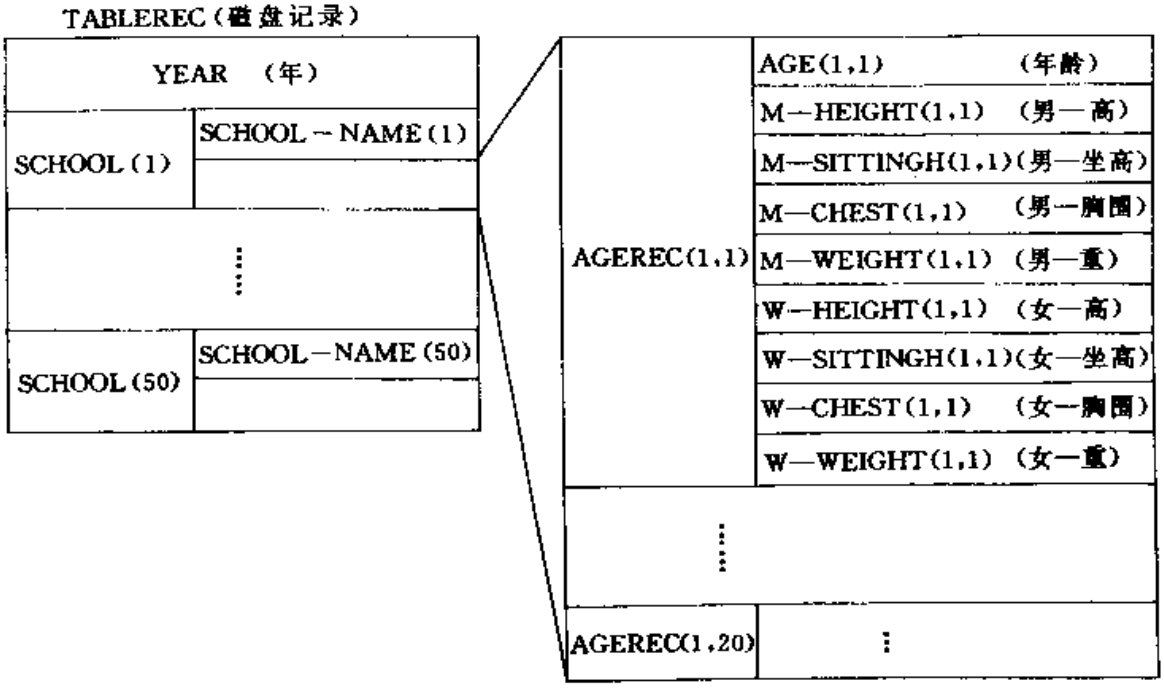



图 9.15

即一个记录中包括某一年的50个学校的情况。每个学校的栏中包括学校名（SCHOOL-NAME）和按20个年龄统计的身体数据AGEREC（包括某一年龄学生的男生和女生的平均

身高、坐着的平均高度、平均胸围、平均体重)。譬如15岁的学生，男生平均身高、坐高、胸围、体重，女生的身高、坐高、胸围、体重。其它年龄如16岁、17岁……都各有一组相应的数据。按年龄大小升序排列的。

今有一数据文件，每个记录记载一个学校的名字和一个年龄数，要求找出该学校和该年龄的身体数据。

程序如下：

```
IDENTIFICATION          DIVISION.
PROGRAM-ID. EXAM94.
ENVIRONMENT             DIVISION.
INPUT-OUTPUT    SECTION.
FILE CONTROL.
        SELECT  IN-FILE    ASSIGN  TO  IN-FILE.
        SELECT  TABLE-FILE    ASSIGN  TO  WK-FILE.
        SELECT  PRINT-FILE    ASSIGN  TO  P-FILE.
DATA        DIVISION.
FILE    SECTION.
FD    IN-FILE  LABEL  RECORD  IS  STANDARD.
01  IN-REC.
    02  SCHOOL-KEY     PIC  X(10).
    02  AGE-KEY        PIC  9(3).
FD    TABLE-FILE  LABEL  RECORD  IS  STANDARD.
01  TABLE-REC.
    02  YEAR       PIC  9(4).
    02  SCHOOL
          OCCURS  50  TIMES
              INDEXED    BY    IX-SCHOOL.
          03  SCHOOL-NAME  PIC  X(10).
          03  AGEREC
              OCCURS  20  TIMES
              ASCENDING  KEY  IS  AGE
              INDEXED  BY  IX-AGE.
              04  AGE  PIC    999.
              04  M-HEIGHT       PIC  9(3)V99.
              04  M-SITTING      PIC  9(2)V99
              04  M-CHEST        PIC  9(2)V99.
              04  M-WEIGHT       PIC  9(2)V99.
              01  W-HEIGHT       PIC  9(3)V99.
              04  W-SITTING      PIC  9(2)V99.
              04  W-CHEST        PIC  9(2)V99.
              04  W-WEIGHT       PIC  9(2)V99.
FD    PRINT-FILE  LABEL  RECORD  IS  STANDARD.
01  PRINT-REC.
```

```
    02  FILLER  PIC  X.
    02  YEAR-P       PIC  9(4).
    02  SCHOOL-P     PIC  BBX(10).
    02  AGE-P        PIC  BB999.
    02  M-HEIGHT        PIC  B(4)ZZ9.99.
    02  M-SITTING       PIC  BBZ9.99.
    02  M-CHEST         PIC  BBZ9.99.
    02  M-WEIGHT        PIC  BBZ9.99.
    02  W-HEIGHT        PIC  B(4)ZZ9.99.
    02  W-SITTING       PIC  BBZ9.99.
    02  W-CHEST         PIC  BBZ9.99.
    02  W-WEIGHT        PIC  BBZ9.99.
PROCEDURE              DIVISION.
READ-PROC.
    OPEN    INPUT         IN-FILE, TABLE-FILE
            OUTPUT        PRINT-FILE.
    READ    TABLE-FILE
            AT  END  GO  TO  TERM.
SELECT-PROC.
    READ  IN-FILE
             AT  END  GO  TO  TERM.
    SET  IX-SCHOOL  TO  1.
    SEARCH     SCHOOL
       AT  END  GO  TO  ERROR-SCHOOL
       WHEN
          SCHOOL-NAME (IX-SCHOOL) =SCHOOL KEY
          NEXT  SENTENCE.
    SEARCH     ALL     AGEREC
       AT     END  GO  TO  ERROR AGE
       WHEN  AGE (IX-SCHOOL, IX-AGE) =AGE-KEY
          NEXT  SENTENCE.
PRINT-PROC.
    MOVE  YEAR  TO  YEAR-P.
    MOVE  SCHOOL-NAME (IX-SCHOOL)   TO  SCHOOL-P.
    MOVE  AGE (IX-SCHOOL, IX-AGE)   TO  AGE-P.
    MOVE  CORR  AGEREC (IX-SCHOOL, IX-AGE)
          TO     PRINT-REC.
    WRITE     PRINT-REC.
    GO  TO  SELECT-PROC
ERROR-SCHOOL.
    DISPLAY'SCHOOL  NAME  NOT  FOUND',
            SCHOOL-KEY.
    GO  TO  SELECT-PROC.
```

```
ERROR-AGE.
    DISPLAY'AGE  NOT  FOUND', SCHOOL-KEY, AGE-KEY.
    GO  TO  SELECT-PROC.
TERM.
    CLOSE  IN-FILE, TABLE-FILE, PRINT-FILE.
    STOP  RUN.
```

今将输入的学校名字项定为SCHOOL-KEY，年龄项定名为AGE-KEY（表示用它们作“键”去查表）。在数据部中定义表，表名为SCHOOL，包括50个学校的数据。每个学校的数据又包括SCHOOL-NAME（学校名）和20个年龄的身体情况数据。今用SEARCH语句来查找。由于50个学校不是按学校名字的顺序排列的，只能用顺序检索法。SEARCH语句要求用位标法，故在对表SCHOOL的定义中用INDEXED BY IX-SCHOOL指定IX-SCHOOL为位标。而年龄—身体数据是以年龄升序排列的。故可用SEARCH ALL语句进行折半检索。因此在AGEREC数据项的描述体中，OCCURS子句内用了ASCENDING KEY IS AGE。在查AGEREC表时用IX-AGE作位标。

在过程部中，先读入存放学生身体数据的磁盘记录，然后读入要查找的校名和年龄。置位标IX-SCHOOL于初值1，使从SCHOOL表中的第一个元素查起。“SEARCH SCHOOL…”语句的作用是找出磁盘记录中SCHOOL-NAME等于输入的SCHOOL-KEY值的SCHOOL表元素来。此时IX-SCHOOL保持当前值。接着再执行“SEARCH ALL AGEREC…”语句，目的是找出在已找出的SCHOOL（IX-SCHOOL）中年龄AGE的值等于输入的AGE-KEY的值的那一个“年龄—身体”表元素AGEREC。注意，在执行“SEARCH ALL”语句时，IX-SCHOOL已是一个已确定的数值，此语句只是用折半法检索AGEREC表。直到找到该学校指定的年龄为止。此时IX-SCHOOL和IX-AGE都保持当前值。

然后执行PRINT-PROC段。将该校该年龄的学生身体数据传送给打印文件记录区（传送时进行数值编辑），输出。

接着再读第二组校名和年龄。如上重复再做一次……。如果找不到指定的学校或年龄，则转到ERROR-SCHOOL段或ERROR-AGE段显示出错误信息，再读下一组校名和年龄。

由上面可知，SEARCH语句功能是比较强的，掌握它的使用，可以使程序简短、明瞭，执行效率高。用位标法引用表元素主要应用于SEARCH语句中。在初步掌握COBOL的基础知识之后，在编制实际应用的表处理的程序时，可以逐步学习并使用这种方法。

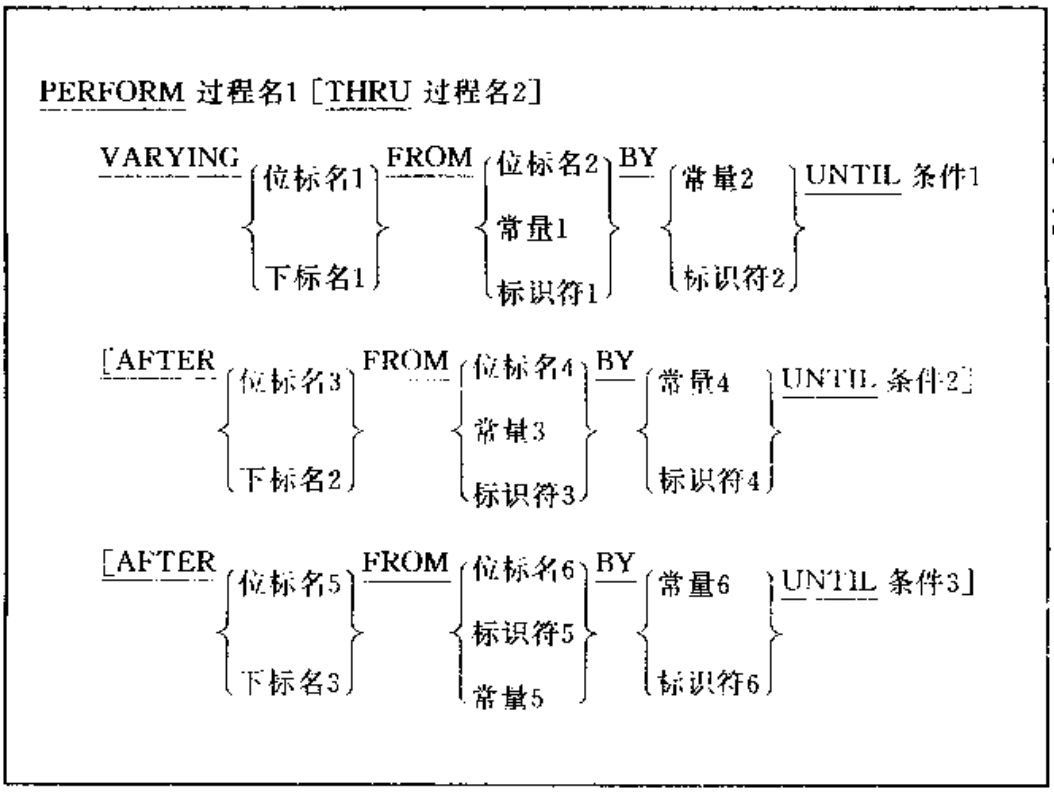
## §9.9 用PERFORM语句对表进行检索

除了使用SEARCH语句检索一个已建立的表外，还可以利用已介绍过的PERFORM语句对表进行检索。

在第六章中学习过下面格式的PERFORM语句，可用来检索一维、二维或三维的表。

这种PERFORM语句可以处理三重循环，而COBOL的表最多也只能三维。因此，可

以用每一层循环来控制表的每一维。COBOL 三维表的元素如 T (A, B, C)，在内存排列的次序是按行存放的，即第三维 (C) 先变化。因此 VARYING 短语这一层循环用来控制三维表的第一维，第二个 AFTER 短语用来控制第二维，最下面的 AFTER 短语用来控制第三维。




**【例9.5】** 假如有一个学生情况的表。在数据部中有如下数据描述：

```
WORKING-STORAGE  SECTION.
77  SCHOOL-IX          PIC  9(2).      (作 SCHOOL 表的下标)
77  CLASS-IX           PIC  9(2).      (CLASS 表的下标)
77  STUDENT IX         PIC  9(2).      (STUDENT 表的下标)
01  TABLE.
02  SCHOOL  OCCURS  10.                (十个学校)
  03  CLASS  OCCURS  5.                (每校五个班级)
    04  STUDENT  OCCURS  40.           (每班40人)
      05  NAME  PIC  X(20).            (姓名)
      05  NUM  PIC  9(5).              (号码)
      05  ADDR  PIC  X(20).            (地址)
      05  AGE  PIC  9(3).              (年龄)
```

在过程部中：

```
    ⋮
PERFORM  SELECTION  THRU  SELECTION-END
    VARYING  SCHOOL-IX  FROM   1    BY    1
            UNTIL  SCHOOL-IX>10
        AFTER  CLASS-IX    FROM  1    BY    1
              UNTIL  CLASS-IX>5
        AFTER  STUDENT-IX  FROM  1  BY  1
```

```
                UNTIL  STUDENT-IX>40.
                :
SELECTION.
    IF  NAME  (SCHOOL-IX, CLASS-IX, STUDENT-IX) ='ZHANG  SHENG'
        AND  AGE (SCHOOL-IX, CLASS-IX, STUDENT-IX) >20
      DISPLAY    STUDENT   (SCHOOL-IX,   CLASS-IX,
                 STUDENT-IX).
SELECTION-END. EXIT.
```

要求找名为“ 'ZHANG SHENG'”，年龄在20岁以上的学生，并输出该生的数据，第一次执行 SELECTION 段，就检查 NAME (1，1，1) 即第一学校，第一班，第一个学生的名字是否“ZHANG SHENG”，年龄是否>20，如是，则输出。然后再第二次执行 SELECTION 段，此时第一个下标 SCHOOL-IX 不变，第二个下标 CLASS-IX 也不变，第三个下标 STUDENT-IX 变成2，这次检查 NAME (1，1，2) 是否等于“ZHANG SHENG”，AGE (1，1，2) 是否>20。如此逐个检查，直到查完此表为止。把所有名为“张生”年龄>20的学生数据都显示出来。

## 习　题

9.1　什么叫表?什么叫表元素?

9.2　按下面的数据描述画出数据结构：

```
01    A.
  02    B  OCCURS    5.
    03    C  OCCURS    3    PIC  X(2).
```

9.3　下面的写法有无错误?

(1)02  T  OCCURS  10  TIMES  PIC  9(3)  VALUE  10.

(2)
```
02  N  OCCURS  5  PIC  X(35).
    03  N1  PIC  X(3).
    03  N2  PIC  X(4).
```

(3)01  TABLE  OCCURS  5  TIMES  PIC  X(80).

9.4　指出哪几个是一维表、二维表或三维表：

```
01  TABLE.
  02  SCHOOL  OCCURS  10.
    03  SCHOOL-NAME  PIC  X(20).
    03  CLASS  OCCURS  10.
      04  CLASS-NAME  PIC  X(10).
      04  CLASS-NUM  PIC  9(4).
      04  STUDENT  OCCURS  30.
      05  STUDENT-NAME  PIC  X(10).
      05  STUDENT-NUM    PIC  9(4).
    03  ADDR  PIC  X(15).
```

9.5　10个工人的工资记录已存放在一个磁盘文件中，每个记录中的数据包括：工人姓

名、工人号码、工人工资。设计一个程序读入这些数据，建立一个包括十个工人工资情况的表，把这十个工人中工资最高的工人名字、工资打印出来，并求十个工人的平均工资。

9.6 有一个产品的价目表如下：

| 产品型号 \ 尺寸 | I | II | III | IV |
|---|---|---|---|---|
| 1 | 10.86 | 11.75 | 12.86 | 13.76 |
| 2 | 5.50 | 7.60 | 9.30 | 10.20 |
| 3 | 17.20 | 18.35 | 19.60 | 20.20 |
| 4 | 15.27 | 16.38 | 17.42 | 18.16 |

定义表的结构如下：

```
02  PRODUCT-NUMBER  OCCURS  4.  (型号)
  04  PRODUCT-PRICE  PIC  99V99  OCCURS  4.
```

请在数据部中对每一型号，每种尺寸的产品按上表赋予初值。

# 第十章　磁带文件和磁盘文件

## §10.1　概　述

我们在前面已经接触到了“文件”这一概念，用到了一些输入输出设备（如键盘、显示器和打印机等）。对一些简单的COBOL程序，只需要用这几种设备就够了。但是在实际应用中，往往要求使用多种输入输出设备。譬如，某些数据（如产品库存的数据，学生成绩数据，职工工资的数据，政府部门历年统计资料等）要求长期存储在磁带和磁盘上，以便随时使用或修改。一个比较复杂的COBOL程序，会经常使用磁带文件或磁盘文件（例如“排序”，是需要用到磁盘或磁带文件的）。在本章中，将简单介绍磁带文件和磁盘文件的某些特性以及COBOL语言是怎样处理磁带文件和磁盘文件的。由于计算机的输入/输出设备不断更新换代种类繁多，我们只能介绍基本的概念和方法，读者可以举一反三灵活应用。

首先要说明，一般所说的“文件”是指外部文件，它是建立在外部介质上记录的集合。在COBOL中对数据的存取是以文件为对象（例如，从磁盘文件中读一组数据，或输出一批数据到磁盘文件中），以记录为单位的（例如，一个WRITE语句输出一个记录到磁盘文件中）。

文件按其介质不同可分为打印文件、卡片文件、磁带文件和磁盘文件等。磁带和磁盘属于可重用介质，即记录在其上的信息可以抹去或修改。如同我们常用的录音磁带可以抹去已录入的歌曲或重录一支新歌一样。在打印机上输出的打印文件是属于不可重用的介质，即打印在纸上的信息是不能抹去或修改的。打印文件的记录长度是固定的（由设备决定），如80或132字符等。而磁带文件，磁盘文件的记录长度是可以变化的。例如一个记录可以是10个字节，也可以是1000个字节，这是由COBOL程序来指定的。

下面简单介绍文件的组织形式和存取方式。

### 10.1.1　文件的组织形式

所谓文件的组织形式是指记录在文件中排列的方式。有以下几种不同组织形式的文件：

(1) 顺序文件 (sequential file)。在建立文件时，先送入的记录排在前面，后送入的排在后面，各记录是连续顺序存放的。以后从文件读记录时，先读出排在最前面的第一个记录，然后是第二个记录……，即先入先出。这和日常生活中排队买东西是相类似的。也就是说，记录在文件中的排列顺序、读出的顺序和建立时的顺序三者是一致的。

(2) 索引文件 (indexed file)。在建立文件时，在存储设备上除了开辟一个区域存放各记录外，还建立一个“目录”以便查找，这个目录表称为“索引”(index)，也称“键文件”(key file)。好象我们写书时，除了正文之外还附加一个“目录”，以后查书时不必从头翻起，只需从目录表上查出所需要找的那一章的开始页数，即可直接从书中找出该章的内容。有关索引文件将在本章第三节中详细介绍。

(3) 直接文件 (direct file)。又叫随机文件，在建立文件时，记录不必顺序存放，由程序指定某一地址直接存放。依次先后送入的两个记录在存储设备上的物理位置不一定是相邻的。读记录时，也必须根据每次给定的键值计算出所需的记录的物理位置再直接取出该记录。

(4) 相对文件 (relative file)。这是 ANSI COBOL 1974 版本新增加的一种文件。在建立相对文件时，除了送入记录本身之外，还要求对记录的顺序编号，如第 1 号记录，第 2 号记录……等，以后可以按记录号直接读出所需的记录。这好比一个班的学生每人有一个座位号 (顺序的)，教师要找第 5 号学生，不必顺序地从第一个学生找起，直接叫第 5 号学生即可。即通过记录号直接查找记录。

不是所有的输入/输出设备都可以使用上述各种文件组织的，只有在磁盘这种大容量存储设备上才能建立索引文件、直接文件和相对文件。而磁带文件和打印文件都只能是顺序文件。

### 10.1.2 文件的存取方式

从一文件中取一个记录或存一个记录到一文件中有以下三种方式：

(1) 顺序存取方式 (sequential access)。每次读 (取) 或写 (存) 的记录总是上次读写的记录的下一个物理位置的记录。如果要读第 100 个记录，必须先读完前 99 个记录才能读第 100 个记录。每次都必须从第一个记录开始读写。这种存取方式简单，但读写费时。如果有 100 个记录，要查找某一记录的平均访问次数为 100/2=50 次。

(2) 随机存取方式 (random access)，或叫直接存取方式 (direct access)。

直接指出记录所在的物理位置来读写所指定的记录，而不必顺序地处理其前面的记录。显然，这种存取方式比顺序存取效率要高。索引文件的目录 (索引)，相对文件的记录号就是专为随机存取的查找而设立的。对直接文件的存取则应直接指出其物理地址。

(3) 动态存取方式 (dynamic access)。它是顺序存取方式和随机存取方式的结合。在本章 § 10.7 介绍。

下表列出文件组织形式和存取的方式的关系。

| 文件组织 | 存取方式 |
|---|---|
| 顺序文件 | 顺序存取 |
| 索引文件 | 顺序存取、随机存取、动态存取 |
| 直接文件 | 随机存取 |
| 相对文件 | 顺序存取、随机存取、动态存取 |

但要说明：索引文件在建立时，不能用随机方式写入，而必须用顺序方式写入，建立后可以随机读写。

读者应当将“文件组织形式”和“存取方式”这两个概念区分开，前者是指记录按什么原则在文件中排列的，后者是指怎样从文件中找到记录。二者有一定联系，但不是一个概念。从存取方式来说只有两大类：不是顺序存取，就是随机存取 (动态存取是这二种存取方式的结合)。没有什么“索引存取”或“相对存取”。只有磁盘文件可以实现随机存取。

因为它是周而复始地旋转的，随便什么时候都可以对指定的某一物理地址中的记录进行存取。而像卡片、磁带等顺序文件，显然不能在读完第5号记录之后再读第3号记录，因为第3号记录已经过去了，不会自动返回。

## §10.2 磁带文件

### 10.2.1 磁带的物理特性

磁带机是用来读/写磁带的专用输入/输出设备。它与家用的磁带录音机的原理和工作方式相似。磁带是一面涂有对磁感灵敏的氧化铁的塑料带，可以在其上记录信息。磁带有许多种，按其宽度分类：主要有1/4英寸，1/2英寸和1英寸三种。按其长度分类有：2400英尺，1200英尺和600英尺等。按其外形分类：有开盘式磁带和盒式磁带两种。目前常用的标准磁带是1/2英寸开盘磁带和1/4英寸盒式磁带。

早期的计算机输入设备主要用卡片机，每张卡片记录80个字符的信息，后来卡片输入逐渐被淘汰。与卡片输入相比用磁带记录数据，有以下明显的优点：

(1) 记录数据的密度较大。初期的磁带每一英寸至少可以记录500～1000个字节，一盘普通的磁带可以记录几十兆字节，有一种海量宽磁带存储器，其容量可达150MB以上。

(2) 读写速度快。磁带的读写速度是由磁带机驱动器的速度和磁带记录密度两个因素决定的。根据磁带机的走带速度，可分为高速磁带机，中速磁带机和低速磁带机。高速磁带机的走带速度约为4～5米/秒，约磁带记录密度为1000字节/英寸，则每秒可读20万个字节。在记录密度相同的情况下，带速越高，传输率就越高。

(3) 便于保存数据。例如可以将大量信息（如图书摘要）存储在磁带上，需要时可调用。

### 10.2.2 磁带记录、块、块间间隙

为了实现读写，磁带上的信息以“块”为单位来存放。每次读写以一个“块”为单位。在两个块之间有一个“块间间隙”。最简单的情况是每一块中只放一个记录。如图10.1所示。

| 记录1 | 间隙 | 记录2 | 间隙 | 记录3 | 间隙 | 记录4 | 间隙 | …… |
|---|---|---|---|---|---|---|---|---|

图 10.1

为了区分各块，块间间隙至少为$\frac{1}{2}$英寸。如果我们以80个字节为一个记录，记录只占1/10英寸的磁带（假设磁带为800字节/英寸），而记录间的间隔为1/2英寸，比记录有效长度大5倍，也就是说，磁带的利用率只有1/6。

为了节约使用磁带，可以在一块中放几个记录，或者说，由几个记录组成一块，如图10.2表示的那样，四个记录组成一块。

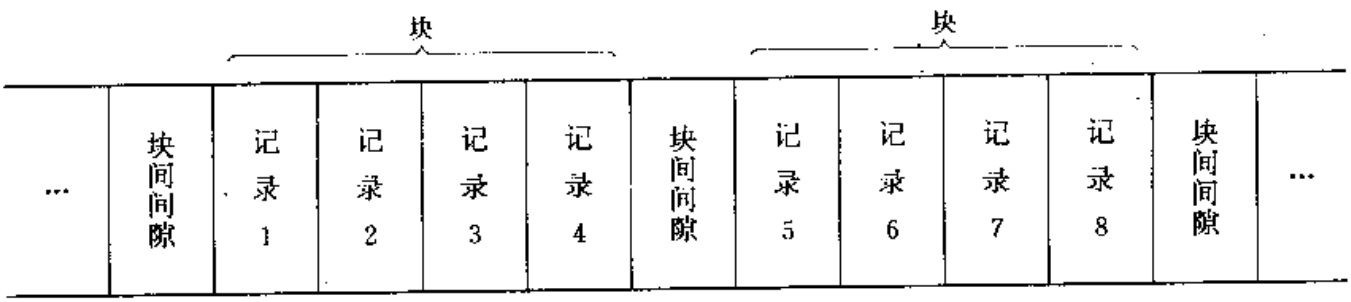



图 10.2

这样，每 4/10 英寸的数据，间隙 1/2 英寸，利用率提高四倍。如果以 100 个记录为一块，则每 10 英寸数据，间隙 1/2 英寸。

我们把块中的记录 1，记录 2，……，称为逻辑记录。块，又称为物理块或物理记录，它是每一次实际存取的单位。每块中包含的记录数称为块化因数。COBOL 程序中处理的记录指的是逻辑记录。而从操作系统的角度来看，是以物理记录（块）为存取单位的。

逻辑记录组块后，如果程序中执行 READ 语句，是怎样动作的呢？请看图 10.3。

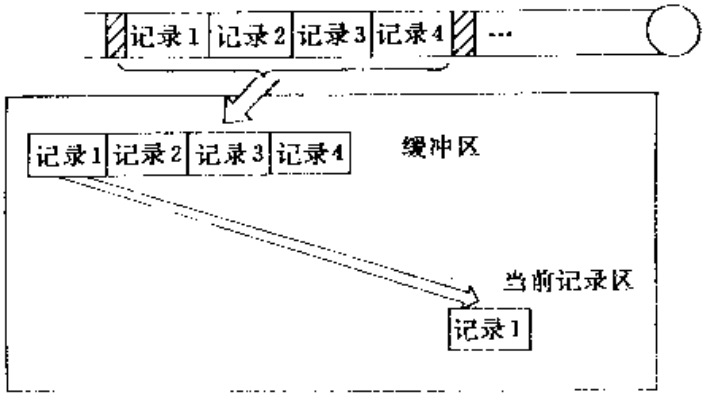



图 10.3

在第一次执行 READ 语句时，计算机从磁带文件中一次将四个逻辑记录读入内存，放在“缓冲区”中，然后将第一个逻辑记录送到“当前记录区”，这个区就是以前学过的“输入记录区”，记录区的内容可供程序使用。当执行第二个 READ 语句时，不再从磁带读数据，而从内存中的缓冲区中读入逻辑记录 2，……直到将四个记录读完为止。如果遇到第五个 READ 语句，则再从磁带读入一块，即逻辑记录 5～8，并将记录 5 送到“当前记录区”，如此重复。

显然，如果一块中包含的记录数愈多，则对磁带文件存取次数愈少。而计算机的输入输出操作是很费时间的。显然，块化因数愈大，则愈能节约计算机时间，但是也不能无限制地加大块化因数，因为必须在内存开辟一个比较大的缓冲区来暂时存放一块中的全部记录。因此，要在节约内存和提高计算机运行效率之间找到一个平衡点。

### 10.2.3 可变长记录

磁带文件中的记录，可以是相同长度的（例如都是 80 个字节），也可以不相同。例如有的记录为 100 个字节，有的为 500 字节。这种变长的记录是管理工作中往往会用到的。譬如，财务科的帐户收支记录。如果在一天之内甲帐户有收支来往 20 次，而乙帐户只有 6 次，

若把一天的收支来往组成一个记录，显然甲帐户的记录长度比乙帐户长很多。那么计算机怎么能判断记录的长度从而将两个记录区分开呢？在COBOL程序中，在数据部中要指定一块包含多少个记录，一个记录包含多少个字节。系统在每个记录的前面增加一个“计数字段”，用来记录本记录中的字节数，见图10.4。

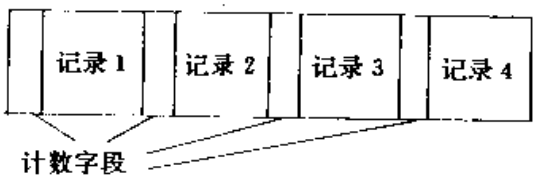



图 10.4

### 10.2.4 磁带文件的组织形式

图10.5表示了一个磁带文件所需的信息项。左面是磁带的开头部分。

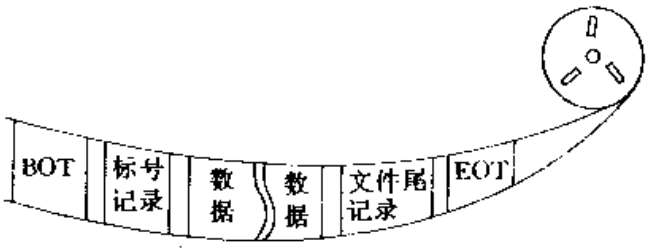



图 10.5

(1) BOT (beginning of tape)。它是“磁带开始标志”的简写，表示带头标记。它是一片贴在磁带上的铝片。在它前面的磁带（约2英尺以上）是不能使用的，只供卷盘缠绕之用。磁带机开始工作时，读写头能自动找出这块标记。

(2) 标号记录 (label record)，也称内部标记 (internal label)。它也是一个数据块，包含下列信息：文件标识（用文件名称或文件号表示）；保留日期（指出该文件应保留到哪一天，在此之前不应向它再写入信息）；建立日期（指出文件建立的日期）；卷序号（如果一个文件有好几卷磁带，称多卷文件，在每卷开头应注明本卷是此文件的第几卷）；等等。

(3) 记录数据块。这是文件的主体部分，用来存放数据记录本身的，一个文件可以包含若干个数据块。

(4) 文件结尾标志。如果一个文件在本卷磁带中结束，则结尾标志为“文件尾”，称带标。如果一文件在本卷磁带没有结束，则标志其为“盘结尾”而文件未结束。这样，将继续处理文件的下一卷磁带。

(5) EOT (end of tape) 也是一铝片，作用是停止磁带驱动器运动以防脱带。当磁带转到EOT位置时，即使程序语句要求磁带再向前走，磁带也不前进。

### 10.2.5 COBOL中有关磁带文件的成分

COBOL中有些成分是专为磁带文件而设的。今集中在本节中说明。

(一) 标识部分。与前完全相同。

（二）环境部

如果文件为多卷文件，则应给该文件分配多台磁带机。在输入输出节中要作说明。

```
INPUT—OUTPUT SECTION.
FILE CONTROL.
    SELECT 文件名 ASSIGN TO 磁带机 1 [，磁带机 2] …
        [；ORGANIZATION IS SEQUENTIAL]
        [；ACCESS MODE IS SEQUENTIAL]
```

如果用三台磁带机，则应分别写出磁带机 1，磁带机 2，磁带机 3 的名字。每种计算机给磁带机定的名字是不完全相同的。例如，IBM 和 M 系列计算机，用 SYS010-UT-S-TAPE 形式，CDC 6000 系列为 TAPE0IA，Honeywell 200 系列为 TAPE UNITMRn。

如果给磁带文件分配了多台磁带机，则一卷磁带处理完后，计算机会自动处理下一卷磁带。如果不指定多台磁带机，则当处理完第一卷磁带后，读写不再继续下去，等待用人工换卷（换下旧卷前还要将卷反绕），换上新卷后才能继续处理。这会大大影响计算机工作效率。

“ORGANIZATION IS SEQUENTIAL”是指出“文件的组织形式是顺序的”。“ACCESS MODE IS SEQUENTIAL”是指出“存取方式是顺序的”，以上两项均可不写。当不写时，隐含表示文件的组织形式是顺序的，存取方式也是顺序的。

（三）数据部

```
FD 文件名
  LABEL { RECORDS ARE } STANDARD
        { RECORD IS   }
  [BLOCK CONTAINS [整数 1 TO] 整数 2 { RECORDS    }]
                                     { CHARACTERS }
  [RECORD CONTAINS [整数 3 TO] 整数 4 CHARACTERS]
  [DATA { RECORD IS  } 数据名 1，[，数据名 2] …]
        { RECORDS ARE }
  [VALUE OF 数据名 3 IS { 数据名 4 } [，数据名 4 IS { 数据名 5 }]…]
                        { 常量 1   }               { 常量 2   }
```

要在数据部中对磁带文件进行描述，说明其标号记录、块化因数、记录长度等。

说明：BLOCK 子句说明一个物理块包含多少个逻辑记录。如果不写 BLOCK 子句，则表示一块中只有一个逻辑记录。如果有：

BLOCK CONTAINS 70 RECORDS

表示一块包含 70 个记录。

如果一个文件中各块中含的记录数不同，可以用以下形式来说明：

BLOCK CONTAINS 50 TO 100 RECORDS

它表示每块记录数在 50 到 100 之间，即块的长度是可变的。在 BLOCK 子句中一般不用 CHARACTERS 代替 RECORDS。

RECORD CONTAIN 子句表示一个记录包含多少字符，其实此子句可以不写。因为在记录的数据描述中已经提供了一个记录的实际长度了。

LABEL RECORD 子句是一定要写的。以前对打印文件用 LABEL RECORD IS OMITTED，表示文件没有标号记录（头标和尾标）。磁带文件一般用 LABEL RECORD IS STANDARD，表示文件按系统规定设立标准标号记录，标号记录中应记入文件名等信息。这个信息是怎样记入的？不同的计算机采取了不同的办法。

（1）在 IBM 和 M 系列计算机中，是用作业控制命令将文件名通知系统的，即它不由 COBOL 程序本身提供。此时，不需要写 VALUE OF 子句。

（2）在某些计算机中，使用 VALUE OF 子句来向标号记录提供以上信息。如在 B6000 系列机 COBOL 中用 VALUE OF IDENTIFICATION IS 'ABC'，表示在文件的标号记录中包括有一个名称为 IDENTIFICATION 的项（标识名项），要求输入文件标识名，今在程序中指定它为"ABC"。而在 Cromemco 中，在 VALUE OF 的后面要求写 FILE-ID，如 VALUE OF FILE-ID IS 'ABC'。不同系统用不同的名字来代表'文件标识名'项。因此，使用 VALUE OF 子句时应查阅所用计算机系统的说明书。

（四）过程部

（1）OPEN 语句

如果打开的是输入文件，系统检查磁带文件的标号记录，看它是否所需的文件。如果打开的是输出文件，则执行 OPEN 语句时，按规定给文件建立一个文件头的标号记录（例如指定一个文件名……）。

磁带文件用到的 OPEN 语句为如下格式：

```
OPEN { INPUT  { 文件名 1 [ { REVERSED                } ] } …
                            { WITH NO REWIND         }       }…
     { OUTPUT { 文件名 2 [WITH NO REWIND] } …                 }…
```

REVERSED 表示"反读"，即从文件的最后一个记录读起，按逆序倒读各记录。但在执行这语句之前，应将文件定位在其尾部。

WITH NO REWIND 表示磁带不反绕。如果不用此可选项，则打开文件时，自动将磁带反转使磁带文件的开头处于磁带机的读写头下面。只有当不需要系统对文件定位时才用此可选项，否则不应该用此可选项。

OPEN 语句的上述可选项用的较少，初学者可不必学它。

（2）CLOSE 语句

只有已经打开的文件才能用 CLOSE 语句关闭。对用 INPUT 方式打开的文件，在 CLOSE 时，检查文件尾标号。对用 OUTPUT 方式打开的文件，在写完输出的记录后关闭时，系统会自动写上文件的尾标。

CLOSE 语句的一般格式为：

```
CLOSE { 文件名 1 [REEL] [ WITH { NO REWIND } ] } …
                               { LOCK      }
```

NO REWIND 的作用和 OPEN 语句中叙述的相同，表示关闭时磁带不反绕。如果不写此可选项，则自动反绕。当需要在同一卷磁带上写两个（或更多）文件时，在写完第一个文件后，不需反绕。

LOCK 可选项用来“锁住”文件，即关闭该文件后，在本程序执行过程中不能再打开，以防止误动作。只有整个程序重新运行，才能再打开此文件。例如：CLOSE A WITH LOCK。

REEL 可选项表示关闭的是某一磁带卷，而不是文件，此时文件仍处于打开状态，直到关闭文件。只有多卷文件才用 REEL 项。

(3) READ 语句

当记录组块时，一次读入内存的是一个物理块而不是一个逻辑记录。

```
READ 文件名 [INTO 数据名] AT END 强制语句
```

(4) WRITE 语句

一般格式为：

```
WRITE 记录名 [FROM 标识符]
```

请注意，对磁带文件的写操作，不应写成下面形式：

```
WRITE REPORT-REC AFTER 2 LINES.
```

因为此时不存在走纸几行的问题。

如果已指定了由四个记录组成一个块，则执行第一个 WRITE 语句时，实际上并不往磁带上写，而是先送到内存中的缓冲区暂存，直到执行四次 WRITE 语句后，缓冲区已放满了，此时才将四个记录整块地输出到磁带上去。记录的拼块（写的时候）和拆块（读的时候）工作，由编译系统的拆拼块子程序专门处理。

由于磁带文件是顺序文件，不能插入新的记录，也不能删除已存在的记录。假如要更新一个磁带文件（例如插入或删除某些记录，修改某些记录中的数据），必须重新建立一个新的文件，将全部记录（包括未修改的和已修改的记录）按顺序写到此新文件中。

### 10.2.6 磁带文件应用举例

**【例 10.1】** 将一批磁盘文件中的数据转储到磁带上，建立一个磁带文件。

假设磁盘文件中的数据是有关产品的数据，其中产品号（占 6 个字节），产品名（15 个字节），单价（6 个字节）。程序可编写如下。

```
IDENTIFICATION  DIVISION.                    (标识部)
PROGRAM-ID.  EXAM10-1.
ENVIRONMENT     DIVISION.                    (环境部)
INPUT-OUTPUT  SECTION.                       (输入-输出节)
FILE-CONTROL.                                (文件控制段)
    SELECT  IN-FILE     ASSIGN  TO           磁盘文件.
```

```
    SELECT  MT-FILE    ASSIGN  TO     磁带文件.
DATA  DIVISION.                       (数据部)
FILE  SECTION.                        (文件节)
FD  IN-FILE  LABEL  RECORD  IS  STANDARD.
    DATA  RECORD  IS  IN-REC.
01  IN-REC.
  02  PRODUCT-CODE(产品号)  PIC    9(6).
  02  PRODUCT-NAME(产品名)  PIC    X(15).
  02  UNIT-PRICE(单价)      PIC    9(4)V99.
FD  MT-FILE  LABEL  RECORDS  IS  STANDARD
        BLOCK  CONTAINS  10  RECORDS
        DATA  RECORD  IS  MT-REC.
01  MT-REC.
  02  PRODUCT-CODE    PIC    9(6).
  02  PRODUCT-NAME    PIC    X(5).
  02  UNIT-PRICE      PIC    9(4)V99.
PROCEDURE  DIVISION.          (过程部)
STA.                          ("开始"段)
    OPEN  INPUT    IN-FILE
          OUTPUT   MT-FILE.
DISK-TAPE.              ("磁盘→磁带处理"段)
    READ    IN-FILE
        AT  END  CLOSE  IN-FILE  MT-FILE
                 STOP  RUN.
        MOVE  CORR  IN-REC  TO  MT-REC.
        WRITE  MT-REC.
        GO  TO  DISK-TAPE.
```

这个程序是很简单的。简单说明如下：

程序中的文件控制段中“文件名”的具体写法请查阅所用的计算机系统的规定。磁带文件和磁盘文件一样，文件的“标号记录”用STANDARD。在过程部中每读入一个记录，即传送到MT-REC（磁盘文件的输出记录区）中。然后写到磁带文件上去。程序中用的是对应项传送（MOVE CORR）。即将IN-REC（磁盘文件的输入记录区）中与MT-REC（磁带文件的输出记录区）中同名的各个数据项一一对应传送。读完磁盘文件的全部记录并传送到磁带文件后，关闭所有文件并停止程序运行。

## §10.3 磁盘存储器的物理特性

磁盘是一有磁感的圆形平面，外形与唱片相似，围绕一个轴等速旋转。磁盘表面有许多同心圆，称为磁道，在它上面记录数据。尽管各个圆的半径不同，但每个磁道上存储容量是相同的。磁盘的最外侧的一个磁道作为0磁道，由外向里顺序编号。见图10.6。前几年流行的IBM-2311系统所用的磁盘每个盘面上约有200个磁道，每个磁道存储容量为

3605 个字节。现在已出现了容量大得多的磁盘存储器，如 M-150 系统用的 F479B 磁盘存储器，一个盘面有 808 个磁道，每个磁道容量为 13165 字节。一个盘面的容量为 10，637，320 字节。

实际上，大多数磁盘存储器不是单个使用的，而是将若干个磁盘装在同一个轴上成为一个磁盘组（disk pack unit）。见图 10.7。最上面的一面和最下面的一面是不使用的，因为它们暴露在外面容易受损。如图所示的磁盘组有五个盘片，共 10 面，能供使用的是 8 面。同一磁盘组的同一磁道号的磁道形成一个圆柱面。例如，每个盘面的 1 号磁道，是一个垂直于盘面的圆柱面，称为 1 号圆柱（cylinder）。如果每个盘片包含 808 个磁道，则整个磁盘组就 808 个圆柱。磁盘组以每分钟约二千多转的速度旋转。磁盘机中有一组可以沿半径方向移动的存取臂，每个存取臂上有一个磁头，用它来读取磁道中的数据或向磁道写数据。如果磁盘组有 8 个可使用的盘面，则应有 8 个读写头，读写头由上而下编号，如读写头 0，1，2，…。

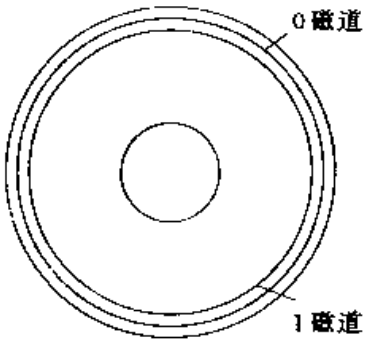



图 10.6

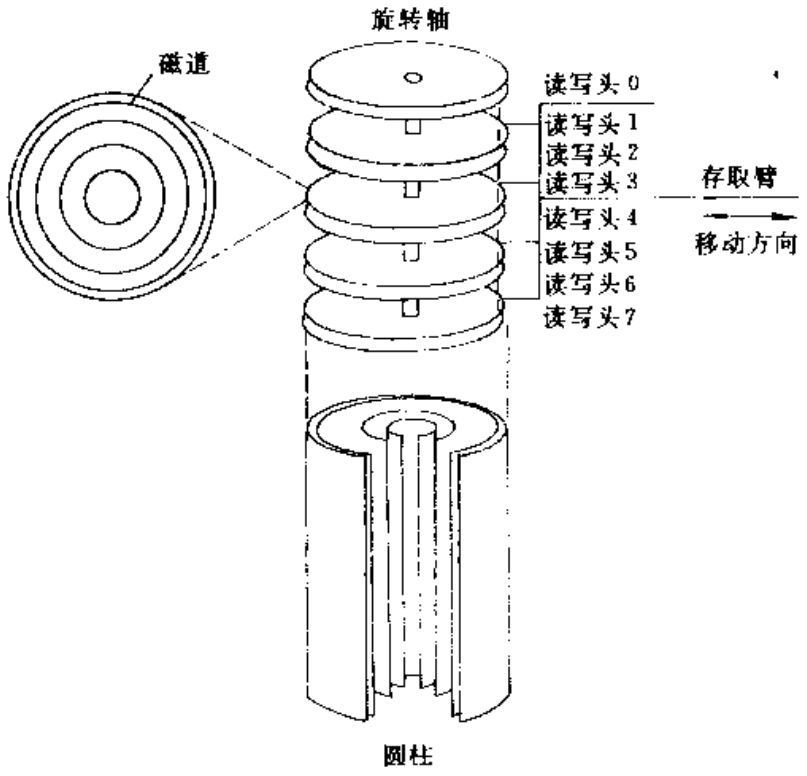



图 10.7

如果要指定一个磁道，从中存取数据，则应给出两个参数，即圆柱号和读写头号。例如，第 5 号圆柱，第 0 个读写头，这时就从第一个可使用的盘面上的第六个磁道（因为柱面号和读写头号均从零开始）中读数据。读某一个磁道中的数据，是这样进行的：读写头移到指定的磁道上等待所需要的记录转到它的下面时，将数据读到计算机内存中，但不会破坏磁盘中原有数据。“写”的情况是相反，但从内存送来的数据代替了磁盘中原有的数据。从指定的磁道中读（写）数据，还需要指出所需的记录是在磁道中什么“位置”，譬如是第几个记录（或第几个物理块）。也可以按“关键字”（KEY）来查找，每个记录中都有一个指定的“关键字”，以便与其它记录相区别。指出它的值就能唯一地确定所需记录的物理位置。有关“关键字”（或译作“键”）的含义将在后面介绍。

如前所述，磁盘文件的组织形式有：顺序文件、索引文件、直接文件和相对文件。使用起来非常灵活。可以任意查询或修改磁盘文件中的数据。磁盘文件在目前计算机系统中得到广泛的使用，尤其在编较大的程序和处理大批数据时，会经常使用它，因此应该很好地掌握和运用它。

由于目前磁盘直接文件使用较少，下面只介绍三种磁盘文件（磁盘顺序文件、磁盘索引文件、磁盘相对文件）。

## §10.4 磁盘顺序文件

按顺序方式组织的磁盘文件与磁带文件非常相似。但比磁带文件用起来灵活些。例如，如果要修改文件中某一个记录，不必重写整个文件。这是因为磁盘是周而复始转动的，已经转过去的记录过一定时间后又会出现在读写头下面，可以对它再处理，而磁带文件则做不到。

### 10.4.1 COBOL 中与磁盘顺序文件有关的成分

使用顺序磁盘文件时，程序中的标识部、环境部、数据部与磁带文件用到的基本相同。过程部则有一些不同。

(1) OPEN 语句

除了可以用 INPUT 方式和 OUTPUT 方式打开文件外，还可以用 I-O 方式，即被打开的文件既可以用来输入，也可以用来输出。

例如：

```
OPEN   I-O   DAFILE.
```

表示 DAFILE 文件是输入-输出文件。请注意，只有磁盘文件可以用 I-O 方式打开，磁带文件是不能既作输入又作输出的。可以利用磁盘文件这一特性，对文件中记录进行更新修改。但应注意，不能用 I-O 方式打开一个尚未建立(即不存在)的文件。有的初学者可能这样想，先往文件上写记录，写完后又把它读出来，这样不是既有输入又有输出吗？于是采用 I-O 方式打开，并写出这样的过程部语句：

```
OPEN   I-O   DAFILE.
  ⋮
WRITE   DAREC.          (DAREC 是 DAFILE 文件中的记录)
READ   DAFILE.
DISPLAY   DAREC.
CLOSE   DAFILE.
STOP   RUN.
```

其原意是先向磁盘文件写数据，再读回来并打印出来。但这样写是错误的。因为 DAFILE 文件原来并不存在，因此无法用 I-O 方式打开。只能先用 OUTPUT 方式打开 DAFILE 文件，往上写数据，然后关闭此文件。再用 I-O 方式打开此 DAFILE 文件，从其中读数据。

不能同时用两种不同的方式打开同一个文件，例如

```
OPEN   INPUT   DAFILE
       I-O     DAFILE.
```

是错误的。

磁盘文件用到的 OPEN 语句一般格式如下：

```
OPEN { INPUT  文件名1 [，文件名2] …
       OUTPUT 文件名3 [，文件名4] …
       I-O    文件名5 [，文件名6] … } …
```

它不像磁带文件那样，没有 NO REWIND 等可选项。

(2) CLOSE 语句

用于关闭磁盘文件的 CLOSE 语句的一般格式为：

```
CLOSE 文件名 [WITH LOCK] [，文件名2 [WITH LOCK]] …
```

例如：

```
CLOSE  FILE1  WITH  LOCK，FILE2.
```

它表示 FILE1 文件关闭后加以“锁住”，FILE2 不锁住。没有 NO REWIND 可选项。

(3) READ 语句

```
READ 文件名 RECORD [INTO 标识符] [；AT END 强制语句]
```

如果执行 READ 语句而文件中已无记录可读，即遇到了文件尾标志，则执行 AT END 子句中的强制语句。

(4) REWRITE 语句

这是用于磁盘文件的专用语句，意为“重写”，用相同长度的记录代替原来的记录，它只能用于由 I-O 方式打开的文件。

在执行 REWRITE 语句之前，必须用 READ 语句从磁盘顺序文件中读出某一记录，可以对该记录内容进行修改，然后用 REWRITE 语句写回原位置上。这样就实现了对磁盘顺序文件中记录的修改。请记住：先读入，后写回。对记录内容可以修改，但不能修改记录的长度和数据的类型。

REWRITE 与 WRITE 不同。WRITE 语句是向磁盘写入一个新记录，而 REWRITE 是用一个新记录代替一个老的记录。

下面是磁盘顺序文件的打开方式与使用读写语句的关系：

| 读写语句 | OPEN 方式 | | |
|---|---|---|---|
| | INPUT 方式 | OUTPUT 方式 | I-O 方式 |
| READ | √ | | √ |
| WRITE | | √ | |
| REWRITE | | | √ |

也就是说，以 INPUT 方式打开的文件只能读，以 OUTPUT 方式打开的文件只能写。以 I-O 方式打开的文件能读和重写。

(5) WRITE 语句

| WRITE 记录名 [FROM 标识符] [; INVALID KEY 强制语句] |
|---|

只适用于在磁盘上建立一个新的顺序组织的文件时用，不能用于修改磁盘文件中的记录。当试图输出的记录超过了分配给文件的空间范围时，就执行 INVALID KEY 子句中的强制语句。

### 10.4.2 磁盘顺序文件应用举例

**【例 10.2】** 磁盘顺序文件 DA-FILE 中已存放了一批货物的库存记录，其数据格式为：

| DAREC | | |
|---|---|---|
| DA PRODUCT-CODE (产品号) | DA-PRODUCT NAME (产品名) | DA-ON-HAND-QTY (库存数量) |
| X (4) | X (20) | 9 (5) |

今输入产品购销情况的记录（记录在磁盘文件 IN-FILE 中），用来修改磁盘的库存记录。磁盘文件 IN-FILE 中记录的数据格式为：

| INREC | | |
|---|---|---|
| IN-PRODUCT-CODE (产品号) | IN-R I CODE (购销代码) | IN-QTY (数量) |
| X (4) | X | 9 (5) |

IN REC 记录中有一个数据项 IN-R-I-CODE，若它的内容为 'R' 时表示“购入”(receipt)。因此，应将此记录中的 IN QTY 项的值（购入的数量）加到原有的该产品的库存数量中去。如果 IN-R-I-CODE 项的值为 'I'，表示“售出”(issue)，应从该产品的库存量中减去 IN QTY。如果此记录中提供的 IN-PRODUCT-CODE 项的值（产品号）在磁盘文件 DA-FILE 中找不到，则表示“无此产品”，打印出“NO PRODUCT”，再处理下一个记录。

两个磁盘文件中的各记录都是已按产品号大小顺序排列好的。

如果磁盘文件 DA-FILE 中第一个记录中产品号为 0001，而磁盘文件 IN-FILE 中第一个记录中产品号为 0002，这表示对磁盘文件 DA-FILE 中 0001 号产品的库存数量不必修改，应读下一个磁盘记录。如果 DA-FILE 的下一个记录的产品号为 0002，则表示应按 IN-FILE 文件提供的数据进行修改（看购销代码 IN-R-I-CODE 的内容是 'R' 还是 'I' 而进行 '加' 或 '减' 的运算）。如果从 DA-FILE 文件读入的记录中产品号不是 0002 而是 0003，则表示 0002 号产品是不存在的，按“出错”处理。

为简单起见，我们假设数据如下：

| DA-FILE 文件记录 | | | IN-FILE 文件记录 | | |
|---|---|---|---|---|---|
| 产品号 | 产品名 | 库存量 | 产品号 | 购销代码 | 数量 |
| 0001 | CAP | 00020 | | | |
| 0002 | SHORES | 00100 | 0002 | P | 00010 |
| 0004 | PEN | 00120 | 0002 | I | 00002 |
| 0006 | PENCIL | 00800 | 0003 | I | 00026 |
| 0007 | CLOCK | 00175 | 0010 | R | 00075 |
| 0010 | CONFECTION | 00035 | 0013 | I | 00023 |
| 0012 | COAT | 00070 | | | |
| 0014 | CHOCOLATE | 00123 | | | |
| 0050 | CELL | 01200 | | | |

程序如下：

```
IDENTIFICATION      DIVISION.
PROGRAM-ID. EXAM10-2.
ENVIRONMENT         DIVISION.
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
      SELECT  DA-FILE    ASSIGN TO DA-FILE
              ORGANIZATION  IS  SEQUENTIAL
              ACCESS  MODE IS SEQUENTIAL.
      SELECT  IN-FILE   ASSIGN TO IN-FILE
              ORGANIZATION  IS  SEQUENTIAL
              ACCESS  MODE IS SEQUENTIAL.
DATA        DIVISION.
FILE    SECTION.
FD  DA-FILE  LABEL  RECORD IS STANDARD
    DATA RECORD IS DAREC.
01  DAREC.
    02  DA-PRODUCT-CODE     PIC X(4).        (产品号)
    02  DA PRODUCT-NAME     PIC X(10).       (产品名)
    02  DA-ON-HAND-QTY      PIC 9(5).        (库存数)
FD    IN-FILE  LABEL  RECORD  IS STANDARD
        DATA  RECORD IS INREC.
01  INREC.
    02  IN-PRODUCT-CODE     PIC X(4).
    02  IN-R-I-CODE     PIC X.
    02  IN-QTY     PIC 9(5).
PROCEDURE          DIVISION.
START-RUN.
        OPEN  INPUT     IN-FILE
```

```
                I-O          DA-FILE.
    REAN   DA-FILE
        AT END MOVE HIGH-VALUE   TO DA-PRODUCT-CODE.
    READ   IN-FILE
        AT END MOVE HIGH-VALUE   TO IN-PRODUCT-CODE.
    PERFORM   RECORD PROCESSING
                UNTIL   IN-PRODUCT-CODE   =   HIGH-VALUE.
    CLOSE   IN-FILE DA-FILE.
    STOP   RUN.

RECORD-PROCESSING.
    IF   DA-PRODUCT-CODE < IN-PRODUCT CODE
        PERFORM   SKIP
    ELSE
        IF DA PRODUCT-CODE   =   IN PRODUCT CODE
          PERFORM MODIFY
        ELSE
          PERFORM ERR.
SKIP.
    READ   DA-FILE
         AT END MOVE HIGH VALUE   TO DA-PRODUCT-CODE.
MODIFY.
    IF   IN-R-I-CODE   =   'R'
        ADD   IN-QTY TO DA-ON-HAND-QTY
    ELSE
        SUBTRACT   IN-QTY   FROM   DA-ON-HAND QTY.
    REWRITE   DAREC.
    READ   IN FILE
        AT END MOVE HIGH-VALUE   TO IN-PRODUCT-CODE.
ERR.
    DISPLAY 'HAVE NOT THIS PRODUCT',INREC.
    READ   IN FILE
        AT END MOVE HIGH-VALUE   TO IN-PRODUCT CODE.
```

程序的流程如图 10.8 所示。

对程序的说明：

今建立了两个文件：输入文件 IN-FILE 和输入输出文件 DA-FILE。根据题目给出的数据格式，在数据部中对输入文件 (IN-FILE) 记录和输入输出文件 (DA FILE) 记录作了数据描述。这些是很容易看懂的。

下面我们对照着流程图分析程序的过程部。

在打开文件后，先读入 DA-FILE 文件记录，再读入 IN-FILE 文件记录。在一般情况下。开始时是不会马上遇到文件结尾标志的，即不执行 AT END 字句（我们在后面再来说明 AT END 子句的含义）。只要 IN-PRODUCT CODE 的值不等于 HIGH-VALUE (IN-FILE

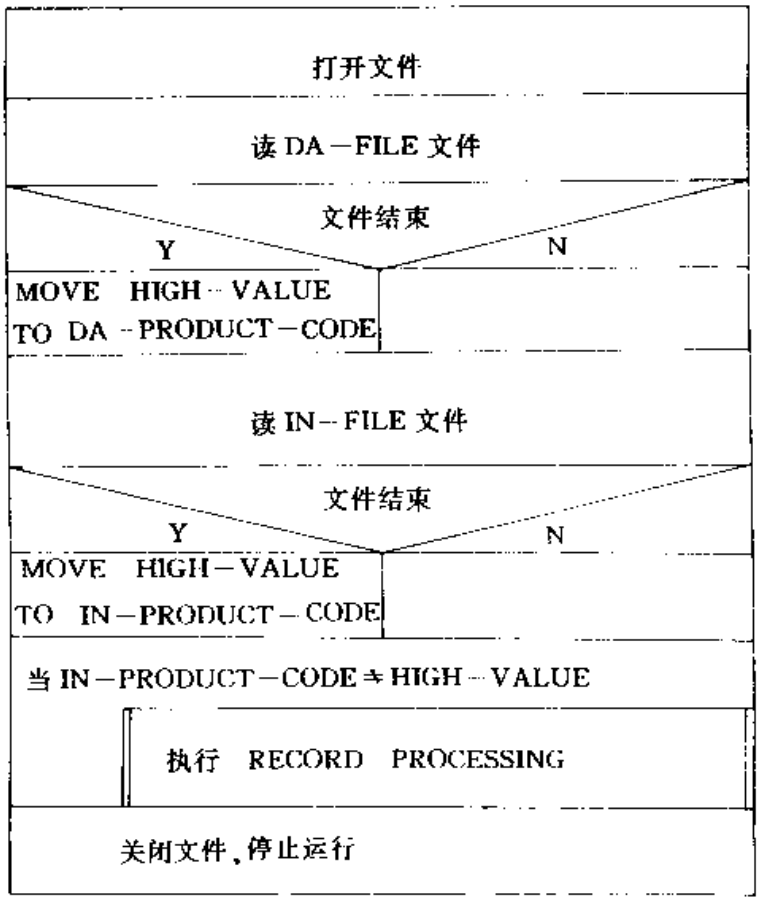



图 10.8

文件记录未读完时，IN-PRODUCT-CODE 的值就不会等于 HIGH-VALUE），就应该按题目要求将 IN-FILE 的记录和 DA-FILE 的记录中的产品号（PRODUCT CODE）进行比较。比较的结果无非三种可能：DA-FILE 记录中的产品号大于或等于或小于 IN-FILE 记录中的产品号。然后分别情况进行处理。

比较和处理的工作由 RECORD-PROCESSING 段来完成。即每新读入一个记录后，都要把两个文件记录中的产品号进行比较并处理。因此要多次执行 RECORD-PROCESSING 段，直到 IN-FILE 的记录全部读完并处理完为止。

我们接着分析 RECORD-PROCESSING 段，请看图 10.9。如果 DA-PRODUCT-CODE

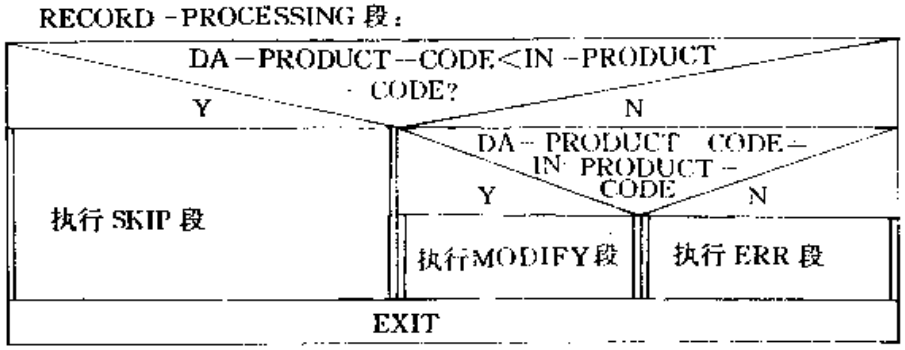



图 10.9

<MT-PRODUC T-CODE，即读入的 DA-FILE 记录中的产品号小于 IN-FILE 记录中产品号（见图 10.10），则不应修改此磁盘记录。应读下一个磁盘记录，这个工作由“SKIP”段来完成。请看框图（图 10.11）。

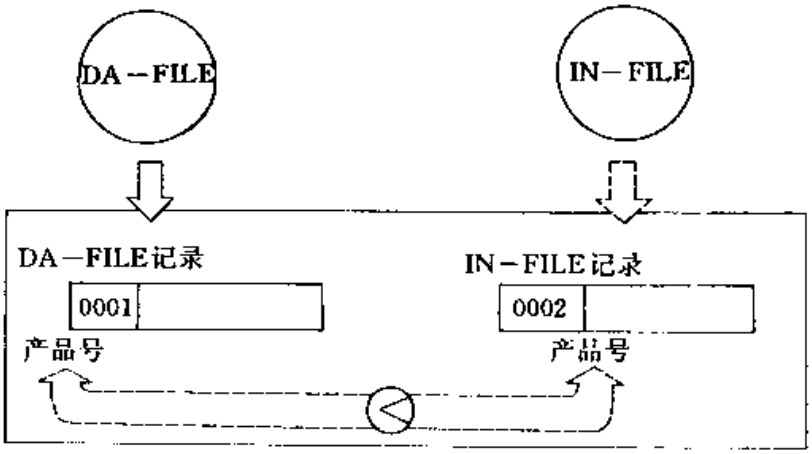



图 10.10

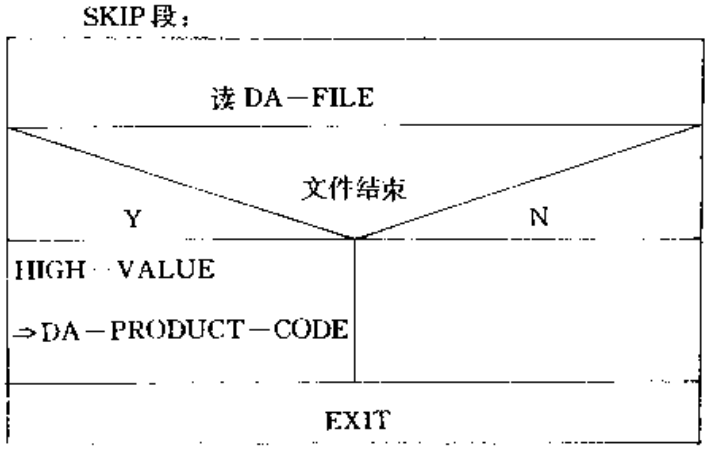



图 10.11

如果 DA-PRODUCT-CODE＝IN-PRODUCT-CODE。则应修改磁盘记录，见图 10.12。这个工作由“MODIFY”段来完成。请看“MODIFY”段的流程图（图 10.13）。

如果 IN-FILE 记录中的 IN-R-I-CODE 的内容为‘R’。则应将库存的数量 DA-ON-HAND-QTY 加上“购入”的数量 IN-QTY。作为该产品新的库存量。如果不是‘R’，就是‘I’，则应从 DA-ON-HAND-QTY 中减去 IN-QTY。将修改过的磁盘记录的内容用 REWRITE 语句重新写回到 DA-FILE 文件上。用以代替原来的库存记录。此后，应再处理下一个 IN-FILE 文件的记录了。所以再读入一个 IN-FILE 文件的记录。为什么不马上再读入一个新的 DA-FILE 记录呢？因为有可能下一个 IN-FILE 记录仍然要修改刚才已修改过的 DA-FILE 记录。譬如，上一个 IN-FILE 记录上的数据为 0002R00010（购入 10 个），下一个 IN-FILE 记录上的数据为 0002I00002（销售 2 个），都要对产品号为 0002 的 DA-FILE 记录进行修改。因此，在处理完上一个 IN-FILE 记录后，该记录的任务已完成，应读下一个 IN-FILE 记录，而 DA-FILE 记录的任务尚未完成，还要看下一个 IN-FILE 记录是否要对其进行修改。

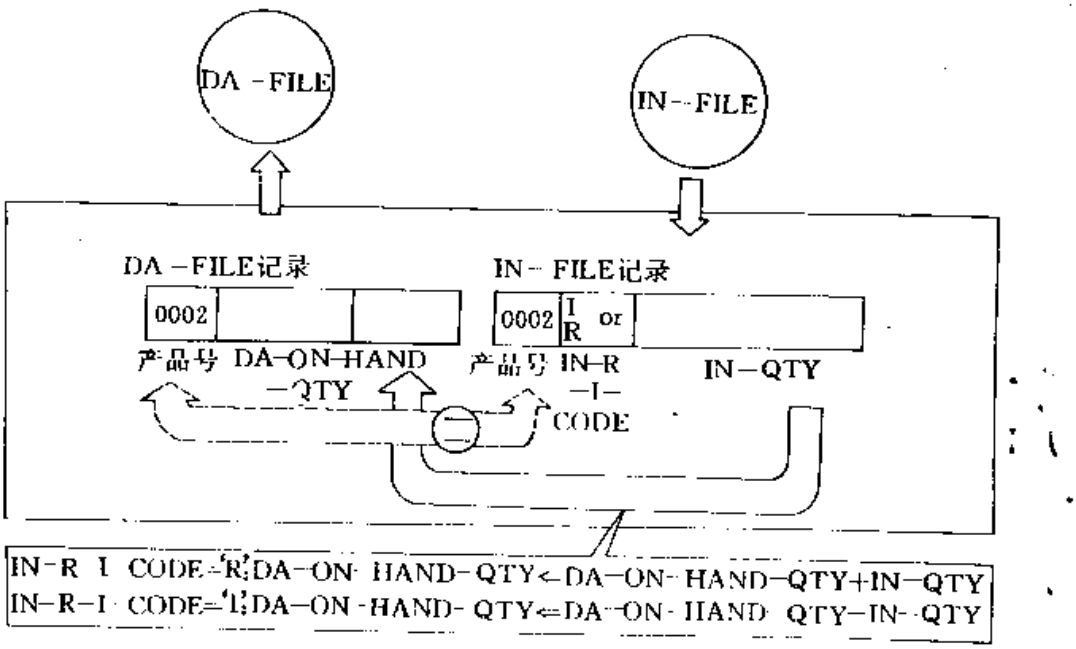



图 10.12

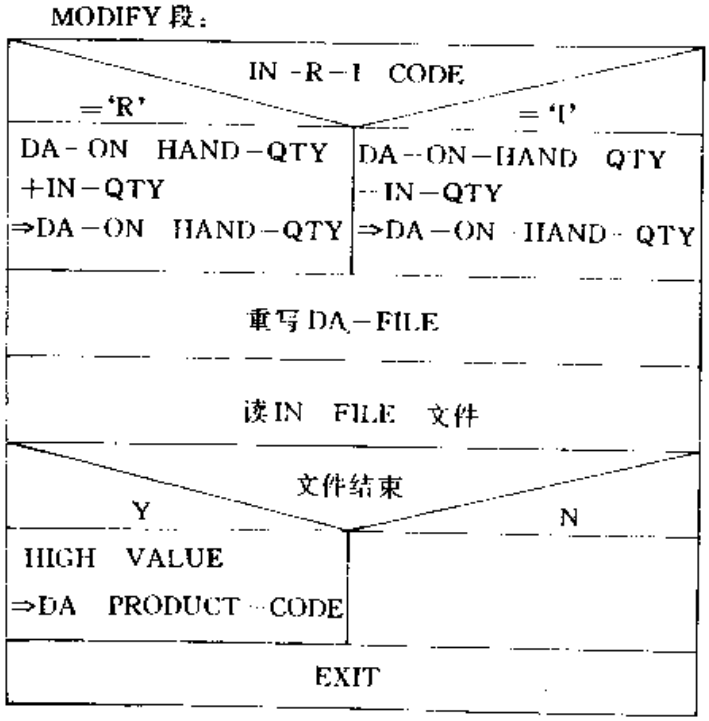



图 10.13

请读者特别注意各段的出口处理。图 10.13 中出口 (Exit) 应转至何处？由于 MODIFY 段是图 10.9 中的一个处理框，因此，执行完 MODIFY 段后应转回图 10.9 中三个处理框的下面，即已执行完了 RECORD-PROCESSING 段。再看图 10.8，执行完一次 RECORD-PROCESSING 段后应进行一次判断操作，如果刚读入的 IN-FILE 记录中的 IN-PRODUCT-CODE 不等于 HIGH-VALUE，则还要再继续调用 RECORD-PROCESSING 段，又要将 DA-PRODUCT-CODE 与 IN-PRODUCT-CODE 相比。注意，此时的 DA-PRODUCT-CODE 的内容未变，仍为上一次的 DA-PRODUCT-CODE 而 IN-PRODUCT-CODE 的内容

已变了，是刚读入的一个 IN-FILE 记录中的产品号。如果两个文件记录的产品号相等，则应再修改一次。如果读入的 IN-FILE 上的产品号大于 DA-FILE 上的产品号，则表示不再修改 DA-FILE 记录。应再读下一个 DA-FILE 记录，并与 IN-PRODUCT-CODE 比较。直到找到一个与 IN-PRODUCT-CODE 相等的 DA-PRODUCT-CODE 为止。譬如 IN-PRODUCT—CODE 为 0010，则 DA-PRODUCT-CODE 值为 0002，0003，……直到 0009 的磁盘记录都不必修改，一个一个地读入、判断，并执行“SKIP”段。

如果 IN-FILE 上产品号 IN-PRODUCT-CODE 为 0003。而 DA-FILE 文件中只有 DA-PRODUCT-CODE 值为 0002 和 0004 的记录而没有 0003 的记录，则读完 0002 的记录后，再读入 0004 的记录。此时 DA-PRODUCT-CODE（值为 0004）大于 IN-PRODUCT-CODE（值为 0003），说明找不到 0003 这样一个产品号（因为在 0004 后面不再可能有 0003 了），作“出错”处理，由程序中的“ERR”段处理。见图 10.14 和 10.15，显示出 have not this product（无此产品）和此 IN REC 记录。然后再读下一个 IN-FILE 文件的记录。

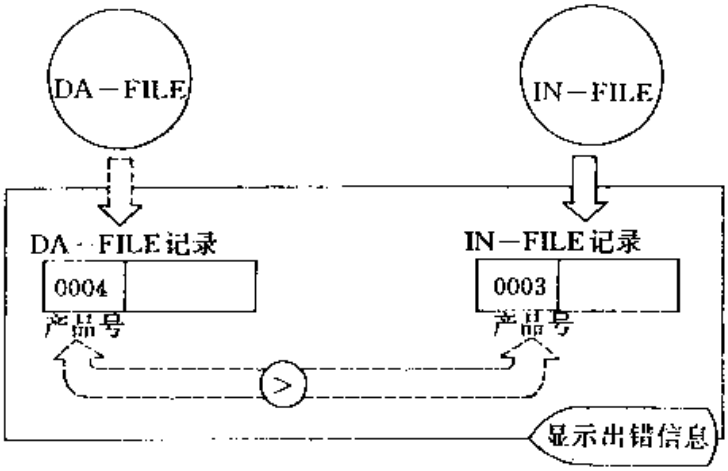



图 10.14

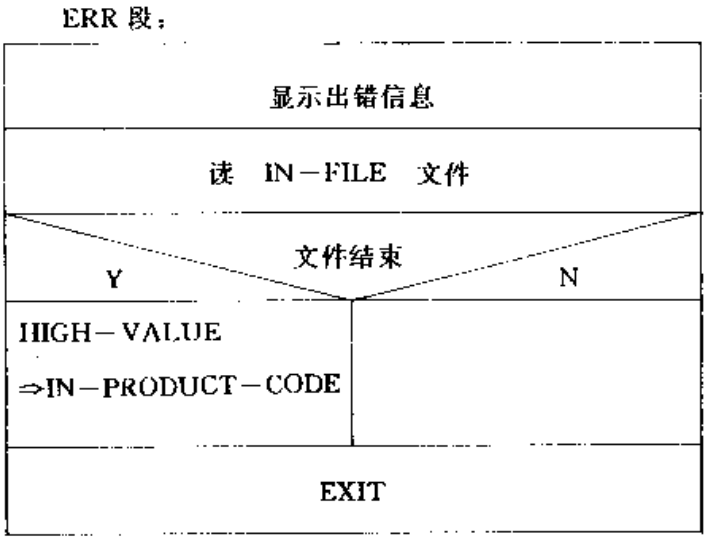



图 10.15

最后我们来看一下当读完了所有 IN-FILE 记录时发生的情况。此时按 AT END 子句要求，应将 HIGH-VALUE 送到 IN-PRODUCT 中去。HIGH-VALUE 是表意常数，它的

值为十六进制的“FF”，即所有二进位上全为“1”，这是最高的值。从图 10.8 可以看出，此时不会再执行下一次 RECORD-PROCESSING 段了。从程序中也可以看到已满足 PERFORM 语句中的 UNTIL 条件，不再执行 RECORD-PROCESSING 段了。关闭文件，结束运行。

如果在读 DA-FILE 文件时遇到文件结束标志，这情况发生在 IN FILE 文件上的产品号大于 DA-FILE 文件上最大的产品号，譬如，DA-FILE 文件的记录中最大的产品号为 0050，而读入的 IN-FILE 记录中产品号为 0060，此时，读完所有的 DA-FILE 中的记录还找不到 0060 的产品号，则应执行 HIGH-VALUE⇒DA-PRODUCT-CODE，即 DA-PRODUCT-CODE 中的值为最大的(所有二进位上全为“1”)。此后的比较必须是 DA-PRODUCT-CODE>IN-PRODUCT-CODE。执行“ERR”段。并一次又一次地读入新的 IN-FILE 记录，直到读完全部的 IN FILE 记录为止。此时 IN-PRODUCT-CODE 中的值为 HIGH-VALUE。不再执行 RECORD-PROCESSING 段。程序结束。

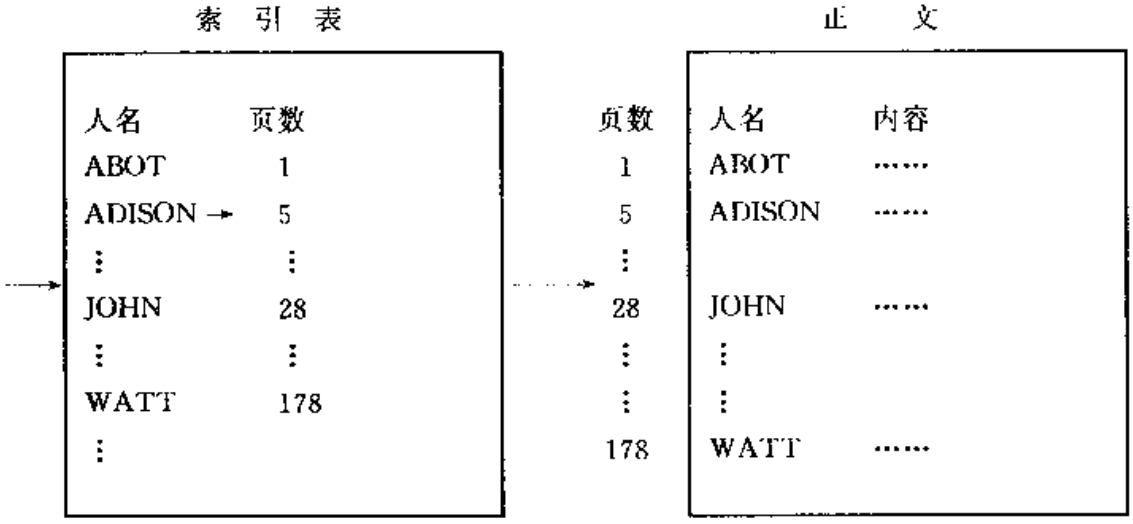
对这个程序我们作了比较详细的说明，以便为以后阅读比较复杂的程序打下基础。希望读者自己能对整个程序整理出清晰的思路。并能根据流程图独立地编出程序。

## §10.5 磁盘索引文件

### 10.5.1 索引文件的概念

顺序文件结构简单，但缺点是查找效率低，如果要找第 1000 个记录，必须从第一个记录开始读，读完 999 个记录以后才能读出第 1000 个记录，显然，这对于提高计算机运算效率是不利的。人们希望能直接找出任何一个所需要的记录来。索引文件、直接文件和相对文件就是通过不同的办法来实现这一目的的。

我们用一个比方来说明索引文件。如果我们编了一本“世界名人简介”的书，是按人名的英文字母的次序编排次序的。为了能迅速地找到所需的人名所在的页数，我们在正文前设立“索引表，印出每一个人名所在的页数。例如要找 ADISON 的简介，先从索引表中找到 ADISON 这一名字，再查出其页数，然后根据页数从正文中直接找出 ADISON 的简介来。显然，它的查找效率要比不设索引表的顺序查找（从第一页开始一页一页地往后找）高得多。




在 COBOL 中，索引文件的概念与此类似。譬如有一批产品记录，它包括以下记录：

| 货　号 | 货　名 | 数　量 | 单　价 |
|---|---|---|---|
| 0001 | CAP | 020 | 0150 |
| 0002 | SHORES | 100 | 1550 |
| 0004 | PEN | 120 | 0350 |
| 0007 | CLOCK | 175 | 1230 |
| 0012 | COAT | 070 | 1835 |

如果我们希望建立一个索引文件，以后用货号作查找的依据（即作“索引”），则我们应先将记录按货号的大小顺序排列好（如上表），并按此顺序将各记录送到计算机中去。在程序中建立索引文件并声明索引文件的索引项是“货号”。这样，在磁盘上建立数据文件（由以上各记录组成）的同时，还会自动在磁盘上设立一个索引表。形式如下表所示：

索　引　表

| 索引项（货号）的值 | 记录在磁盘上的地址 |
|---|---|
| 0001 | 地址 1 |
| 0002 | 地址 2 |
| 0004 | 地址 3 |
| 0007 | 地址 4 |
| 0012 | 地址 5 |

它相当于书的目录。需要寻找某一产品记录时，不必顺序地从磁盘文件中找，而是先按“货号”从索引表中查出此货号所在的记录的地址，然后直接从该地址中找出所需的记录。

这里，我们是以“货号”作查找的依据（即索引）的，掌握“货号”是寻找的前提和关键。我们称“货号”为“关键字”（key word），或简称为“键”（key），意思是解决问题的钥匙。索引表又称为“键文件”，它由一批“关键字”和地址组成。由上述可知，一个磁盘索引文件包括两个文件，即由主文件全部记录构成的数据文件和作为索引用的键文件。

上面我们是以“货号”作为查找时使用的“关键字”（即索引项）。我们也可以不以“货号”而以“货名”作“关键字”（索引项），这时，在建立索引文件时就应该按货名的英文字母顺序将记录排列好，并输出到磁盘上去。记录次序如下：

| 货　号 | 货　名 | 数　量 | 单　价 |
|---|---|---|---|
| 0001 | CAP | 020 | 0150 |
| 0007 | CLOCK | 175 | 1230 |
| 0012 | COAT | 070 | 1835 |
| 0004 | PEN | 120 | 0350 |
| 0002 | SHORES | 100 | 1550 |

这时，索引表不再按货号建立，而是按货名建立了。

索 引 表

| 索引项值 | 记录地址 |
|---|---|
| CAP | AD1 |
| CLOCK | AD2 |
| COAT | AD3 |
| PEN | AD4 |
| SHORES | AD5 |

记录的排列

| | 货 名 | 货号 | 数量 | 价格 |
|---|---|---|---|---|
| AD1 | CAP | 0001 | 020 | 0150 |
| AD2 | CLOCK | 0007 | 175 | 1230 |
| AD3 | COAT | 0012 | 070 | 1835 |
| AD4 | PEN | 0004 | 120 | 0350 |
| AD5 | SHORES | 0002 | 100 | 1550 |

由上可知，用户在使用索引文件时，必须事先指定好以哪一个数据项作为检索的“关键字”。用户选择好“关键字”（索引项）后，怎样通知计算机呢？在写 COBOL 源程序时，用 RECORD KEY（记录键）子句来指定某一数据项作为“关键字”。例如，我们以货号或货名作关键字（索引项），就应该在程序中写：RECORD KEY IS PRODUCT-CODE 或 RECORD KEY IS PRODUCT-NAME。所谓“记录键”，顾名思义，是说明在文件中，是根据这个键的值来安排各记录的逻辑顺序的。

在建立好索引文件后，怎样从索引文件中读出所需要的记录呢？毫无疑问，应该对该“关键字”赋给一个确定的值。例如，如果以货号为“关键字”，则应在读记录前给这个关键字一个值，譬如货号是 0007，或 0012 等。也就是要赋一个具体的值给数据项 PRODUCT CODE。

在 COBOL 程序中，把一个具体的值赋给“关键字”，在不同的计算机系统中采用不同的方法。大致有以下三种不同的方法（读者在使用计算机时应先了解所用的计算机系统采用的是哪一种方法）。

（1）直接给作为 RECORD KEY 的数据项赋值。例如，如果在 RECORD KEY 子句中指定的是 PRODUCT-CODE（货号），在过程部中我们可先将某一货号（例如 0004）传送（MOVE）给“记录键”项 PRODUCT-CODE，再执行 READ 语句，由于此时作为记录键项的 PRODUCT-CODE 的当前值已为 0004，因此读文件时先在索引表中找到 0004 这一索引项所对应的地址 3（见上页），再从地址 3 中读出此记录。PDP-11 等计算机采用此方法。

（2）在建立索引文件时指定 RECORD KEY，而在存取检索时，要另外指定一个 NOMINAL KEY（“名义键”，或称“标定键”）。譬如，指定：

RECORD KEY IS PRODUCT-CODE NOMINAL KEY IS A.

A 是在数据部中已定义的初等项，名字是任意的。如果想读产品号为 0004 这一记录，则在过程部中先将 0004 传送给 A，然后执行 READ 语句，这时系统会按 NOMINAL KEY 所指定的数据项 A 的当前值（0004）到索引表中查找出该记录的地址。也就是说，在向索引文件读写之前，必须给“名义键”项赋一个值，如 MOVE ‘0004’ TO A。

名义键项可在 WORKING-STORAGE 中定义，而且它的 PIC 描述必须和记录键项一样。

（3）有的计算机系统不用 NOMINAL KEY 子句指定存取的键，而用 SYMBO LIC

KEY（符号键）子句。

这三种方法虽然具体规定不同，但基本原理和方法是相同的。读者只要弄清楚一种，就可以举一反三。使用前查一下本系统的说明书即可。

关于索引文件，要指出以下几点：

（1）索引文件分为两类：索引顺序文件（Indexed Sequential file）和索引非顺序文件（Indexed Non-Sequential file）。前者的数据文件是按指定的索引项值的递增顺序排列的。后者的数据文件是可以不按此顺序排列的。而两者的索引区中的键文件都是按上述递增顺序排列的。

在建立索引非顺序文件时，数据记录按物理次序存入，同时系统自动建立索引区，即按照数据顺序记入索引项。在数据全部输入以后，系统再自动将索引区排序。索引表在排序前按照记录的物理顺序，排序后按照记录的关键字的逻辑顺序，而将原索引表抹去。

删除一个记录时，删去相应的索引项，而数据记录保持不变。插入新记录时，将插入的记录置于数据区最后，并将加入新的索引项的索引表（键文件）重新排序。此排序也是由系统自动进行的。

索引顺序文件比索引非顺序文件查找速度高，索引区占空间少。因而索引顺序文件应用很广。由于考虑到效率问题，大型文件几乎都采用索引顺序文件形式。IBM 和 M 系列机提供的也是索引顺序文件。读者在使用计算机时，应先了解该系统提供的是索引顺序文件还是索引非顺序文件。本书所介绍的是索引顺序文件，但它的使用方法基本上都适用于索引非顺序文件。

（2）RECORD KEY 指定的数据项必须是索引文件记录的一部分。例如，上面的例子中，只能以记录中的货名、货号、数量或价格四者之一作记录键。显然不能以不属于此记录的其它数据项（如工厂名）来作 RECORD KEY，否则该怎么排列各记录呢？

（3）建立索引顺序文件时，应该用顺序存取方式，在向磁盘文件写记录时，后一个记录的记录键值必须大于前一个记录的记录键值。如果以货号为记录键，则必须先将货号最小的记录送到文件中去。如果在送了货号为 0004 的记录之后，又送一个货号为 0003 的记录，则破坏了应有的顺序，系统将按出错处理。也就是说，记录键值必须严格递增（而索引非顺序文件可以不必按记录键值递增的次序存入记录）。也不能有两个记录键值相同的记录，否则怎么检索呢？好比一本书的目录有两个第一章，究竟以哪一个作标准呢？

（4）记录键的描述体不能包含 OCCURS 子句，也不能从属于含有 OCCURS 子句的数据项。*例如*：

```
02  PRODUCT OCCURS 10.
  03  PRODUCT-CODE PIC 9(4).
  03  PRODUCT-NAME PIC X(15).
      ⋮
```

若以 PRODUCT CODE 作 RECORD KEY 是不对的。因为它从属于带 OCCURS 子句的 PRODUCT.

（5）RECORD KEY（记录键）和 NOMINAL KEY（或 SYMBOLIC KEY）应当有相同的数据描述。许多系统规定它们的长度不能超过 256 个字符。

（6）作为 NOMINAL KEY（或 SYMBOLIC KEY）的数据项不能出现在本文件的记录

中。如上例中，不能以货名、货号、数量、价格中的任一个为 NOMINAL KEY（或 SYMBOLIC KEY），否则就会破坏该记录了。它可以是在 WORKING STORAGE 节或其它文件的记录中定义的数据项。

如：在 WORKING-STORAGE 节中对 A 作了定义：

```
WORKING-STORAGE SECTION.
77  A PIC 9(4).
```

A 的描述为 PIC 9(4)，如果作为 RECORD KEY 项的数据项 PRODUCT CODE 的描述也是 PIC 9(4)则二者的描述是一样的，这样是正确的。

索引文件中的记录是顺序存放的，因此也可以顺序存取，即从头或某一记录开始一个一个地读写。这时可不必指定 NOMINAL KEY（或 SYMBOLIC KEY），而用下面介绍的 START 语句来确立开始顺序读写的记录。

### 10.5.2 COBOL 中与索引文件有关的成分

（一）标识部。无特殊要求，与以前一样。

（二）环境部

```
SELECT 文件名 ASSIGN TO 磁盘机名
    ORGANIZATION IS INDEXED
    [; ACCESS MODE IS {SEQUENTIAL / RANDOM}]
    RECORD KEY IS 数据名 1
    [NOMINAL KEY IS 数据名 2] .
```

说明：

(1) ORGANIZATION IS INDEXED 是说明文件组织形式是索引文件，这是 ANSI COBOL 1974 文本要求的。在某些计算机系统中可不写此项，例如在 IBM360 和 M150 等计算机中，在其磁盘机名中已包含了对文件组织形式的说明，如磁盘机名写为：SYS010-DA-I-DISK，其中第三项 'I' 就是代表“索引文件组织”（如果写“S”是顺序文件，写“D”代表直接文件）。

(2) ACCESS MODE IS RANIDOM 说明存取方式是随机的。如果写 ACCESS IS SEQUENTIAL 则表示顺序存取。如不写 ACCESS 子句，则隐含为顺序存取方式。

(3) RECORD KEY IS 数据名 1 是指定用数据名 1 作为“记录键”，如前所述，数据名 1 应该是索引文件记录中的一项。

(4) NOMINAL KEY IS 数据名 2 是指定用数据名 2 作为“名义键”。ANSI COBOL 1974 文本中无此子句的规定。但目前使用的不少计算机是有这一子句的（存取时用 NOMINAL KEY 或 SYMBOLIC KEY 来查找索引表）。如果是初次建立一个新的索引文件而不去查找读写，则不必指定 NOMINAL KEY。

（三）数据部　作为 RECORD KEY 和 NOMINAL KEY 的数据项应在数据部定义。

（四）过程部

(1) OPEN 和 CLOSE 语句形式与磁盘顺序文件相同。可以用 INPUT，OUTPUT 或 I-

O 三种方式打开文件。

(2) READ 语句

READ 文件名 RECORD [INTO 数据名] [; INVALID KEY 强制语句]

随机读一记录，必须先向 RECORD KEY（或 NOMINAL KEY 或 SYMBOLIC KEY，视计算机系统不同而定）提供一个值。在执行 READ 时，就根据该键的值从索引表中找到记录地址，从该地址中读出记录。如果所提供的键值在索引表中找不到（譬如指定要找产品号为 0027 的记录，而索引表中只有 0028 或 0029 的产品号），则认为所提供的键值是无效的，不执行 READ 指令，而执行 INVALID KEY 子句中所指定的强制语句。

应当注意。在随机读写中，READ 和 WRITE 语句中不用 AT END 子句，而用 INVALID KEY（无效键）子句。因为只有顺序读写才会遇到文件结束标志，出现 AT END 条件。随机读写时是指向某一特定的记录，不是顺序查找，不应指定 AT END 子句。

假如在某一程序中，已指定 PRODUCT-CODE 为记录键，A 为 NOMINAL KEY（例如 M150 等系统 COBOL 要求指定 NOMINAL KEY）。则可以：

```
MOVE 12 TO A.
READ PRODUCT-FILE INVALID KEY
    CLOSE PRODUCT-FILE  STOP RUN.
```

读出 PRODUCT-CODE 为 0012 的记录，如果不存在此记录，则关闭文件，停止运行。

索引文件也可以顺序读，用：

READ 文件名 AT END 强制语句

从第一个记录读起，和读顺序文件的方法相同。也可以从当中某一记录读起，用 START 语句指定始读的位置。但顺序读时应在设备部中说明 ACCESS MODE IS SEQUENTIAL。

(3) START 语句

START 语句只用于顺序读一个索引文件时指定顺序读的起点。

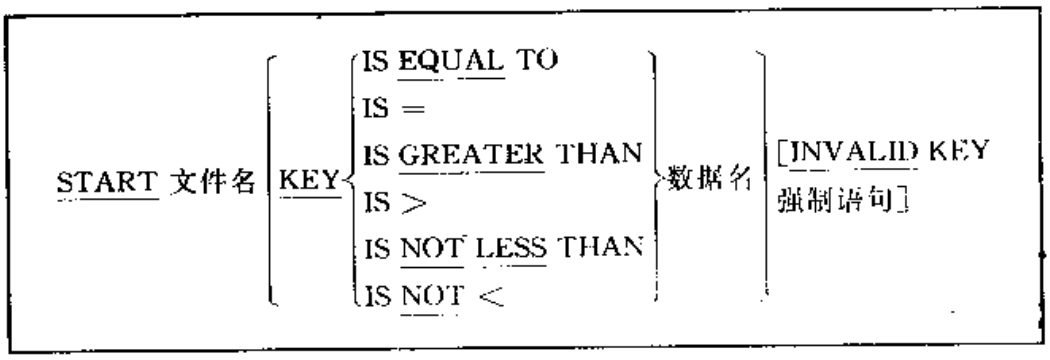
如果有一批国家的记录，已指定 COUNTRY-NAME（国家名）为 RECORD KEY（记录键）。今希望把字母 T 打头的和 T 以后字母打头的国家的记录读出来。索引文件名为 COUNTRY-FILE，过程部中语句为：

```
    ⋮
MOVE 'T' TO COUNTRY-NAME.
START COUNTRY-FILE
    KEY IS NOT<COUNTRY-NAME
    INVALID KEY GO TO ERR.
READ COUNTRY-FILE
    AT  END CLOSE COUNTRY-FILE STOP RUN.
```

START 语句的作用是将记录指针指向索引表中国家名为 T 开头的那个记录。下次执行 READ 语句时，就从这个记录顺序往下读。如果索引表中所有国家名其打头的字母最高为 P，则找不到国家名字以 T 或以 T 以后字母打头的记录，执行 START 语句中的 IN-

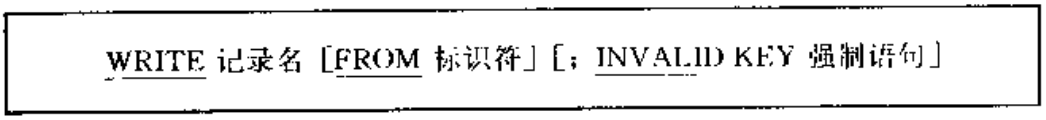
VALID KEY 子句。

START 语句一般格式：

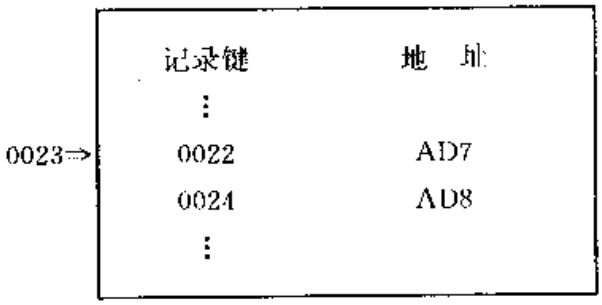



上面一般格式中的“数据名”应是已被指定为“记录键”的数据项名。如上例中的COUNTRY-NAME。

(4) WRITE 语句




如果已建立好了一索引文件，记录是按记录键值的顺序排列的。假如想增加一个记录，例如想在 0022 和 0024 之间插入一个 0023，则先将 0023 传送给 RECORD KEY（或 NOMINAL KEY 或 SYMBOLIC KEY，视不同计算机系统而不同）。然后再用 WRITE 语句写。则系统会将 0023 记录插入到 0022 记录和 0024 记录之间，即追加一个记录，并在索引表中相应地加了一项。如果索引表中原来已有一个记录键值为 0023，则不能插入，执行 INVALID KEY 子句中的强制语句。




也可以用 WRITE 语句建立一个新的索引顺序文件。此时可以指定顺序存取方式，记录应按记录键值递增排列。例如以国家名（COUNTRY-NAME）为记录键，第一个写到磁盘索引文件的记录为：CHINA……

第二个记录为

KOREA…

是可以的，而第三个写的记录如果是：

JAPAN…

则出错。因 J 比 K“小”。也不能先后二次写入记录键值相同的项。否则执行 INVALID KEY 子句。

用 WRITE 语句只能向索引文件写入一个新记录，而不能更新（重写）一个记录。

(5) REWRITE 语句

用来重写一个记录，即用一个新记录代替文件中一个原来的记录。只能在以 I-O 方式打开的文件中使用。

REWRITE 记录名 [FROM 标识符] [; INVALID KEY 强制语句]

在随机读写时先给 RECORD KEY（或 NOMINAL KEY 或 SYMBOLIC KEY）一个值。然后执行 REWRITE 语句，系统会按记录键的值找到所需的记录，以新写入的记录代替原记录。如果找不到与之匹配的记录，则执行 INVALID KEY 子句。

当索引文件的存取方式定义为顺序方式时，为了修改一个记录，需要先读入该记录，然后在内存中修改此记录，并用 REWRITE 语句将修改后的记录重新写入文件以代替原记录。如果记录键值被修改，即内存中修改后的记录的记录键值不再等于刚读出的记录的记录键值，则执行 INVALID KEY 子句。

(6) DELETE 语句

DELETE 文件名 RECORD [; INVALID KEY 强制语句]

根据 RECORD KEY (在有的系统中规定是 NOMINAL KEY 或 SYMBOLIC KEY) 的值删除索引文件中的一个记录。如找不到与之匹配的记录，则执行 INVALID KEY 子句。

### 10.5.3 索引文件应用举例

**【例 10.3】** 建立一个索引文件。

从磁盘数据文件读入一批记录。要求以产品号作记录键，在磁盘上建立一个索引顺序文件。

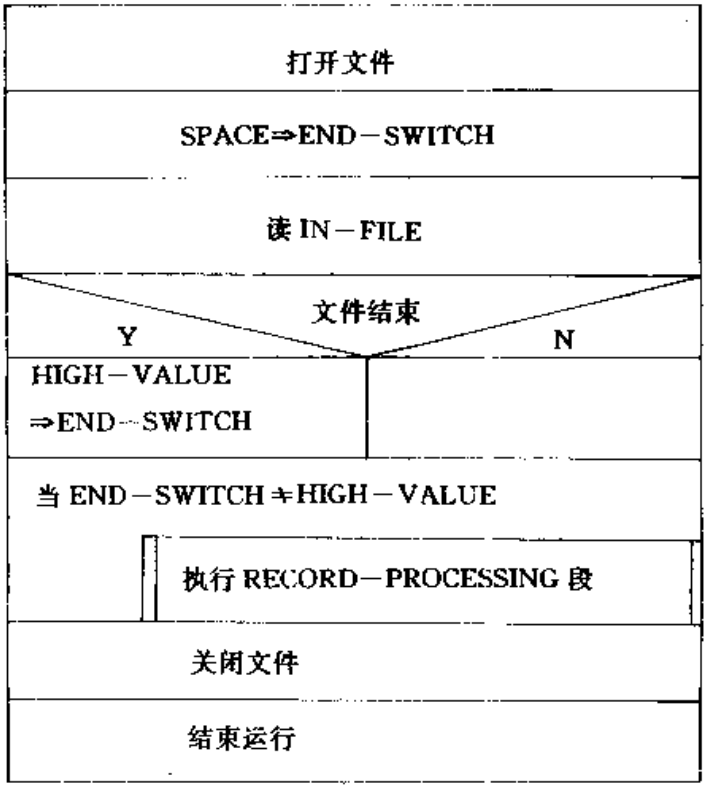
程序为：

```
IDENTIFICATION    DIVISION.
PROGRAM-ID. EXAM10-3.

ENVIRONMENT       DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
      SELECT  IN-FILE   ASSIGN TO IN-FILE.      (输入文件名)
      SELECT  DA-FILE   ASSIGN TO INDE.         (索引文件名)
              ORGANIZATION  IS  INDEXED
              ACCESS  MODE IS SEQUENTIAL
              RECORD KEY IS DA-PRODUCT-CODE.

DATA      DIVISION.
FILE  SECTION.
FD    IN-FILE  LABEL  RECORD IS STANDARD
      DATA RECORD IS CORREC.
```

```
01 INREC.
   02  IN-PRODUCT-CODE     PIC X(4).
   02  IN-PRODUCT-NAME     PIC X(10).
   02  IN-UNIT-PRICE       PIC 9(7).
FD    DA-FILE  LABEL  RECORD  IS STANDARD
         DATA  RECORD IS DAREC.
01 DAREC.
   02  DA-PRODUCT-CODE     PIC X(4).
   02  DA-PRODUCT-NAME     PIC X(10).
   02  DA-UNIT-PRICE       PIC 9(7).
WORKING-STORAGE SECTION.
77 END-SWITCH  PIC X (4).

PROCEDURE       DIVISION.
START-RUN.
       OPEN  INPUT     IN-FILE
             OUTPUT    DA-FILE.
       MOVE SPACE TO END-SWITCH.
       READ  IN-FILE
          AT END MOVE HIGH-VALUE  TO END-SWITCH.
       PERFORM  RECORD-PROCESSING THRU RECORD-EXIT
                UNTIL  END-SWITCH = HIGH-VALUE.
       CLOSE  IN-FILE DA-FILE.
       STOP  RUN.
RECORD-PROCESSING.
       MOVE  IN-PRODUCT-CODE  TO  DA-PRODUCT-CODE.
       MOVE  IN-PRODUCT NAME  TO  DA-PRODUCT-NAME.
       MOVE  IN-UNIT-PRICE  TO  DA-UNIT-PRICE.
       WRITE  DAREC  INVALID  KEY
              DISPLAY 'INVALID  KEY  INPUT=', DAREC.
       READ  IN-FILE  AT  END
              MOVE  HIGH-VALUE  TO  END-SWITCH.
       RECORD-EXIT.  EXIT.
```

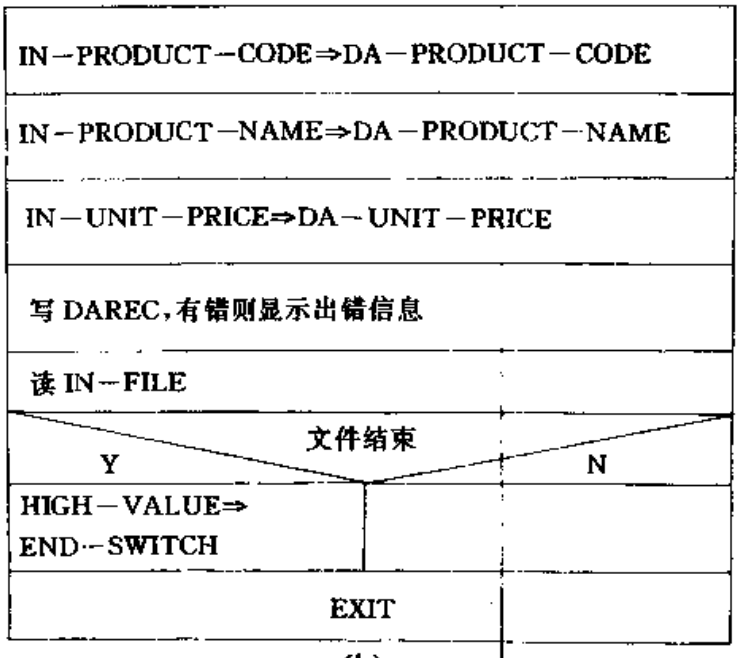
这个程序的流程图见图 10.16，其中（b）图是 RECORD-PROCESSING 段的流程图。区域图见图 10.17。这个程序是比较简单的。有了以前看程序的基础，分析这个程序是很容易的。

今只作一些简单的说明：在程序环境部中用 ORGANIZATION 子句来指定文件组织形式。由于是建立索引顺序文件，因此存取方式是顺序的而不应写成 ACCESS MODE IS RANDOM。记录键指定为 DA-PRODUCT-CODE，即以产品号为索引项，记录按产品号递增顺序排列。必须保证输入的 IN-FILE 的记录是以产品号递增顺序排列好的，否则就会在执行 WRITE 语句时发生错误而导致执行 INVALID KEY 子句。请注意，RECORD KEY 不应指定为 IN-PRODHCT-CODE，而应是 DA-PRODUCT-CODE。请读者想一想为




(a)

RECORD－PROCESSING 段：




(b)

图 10.16

什么？

每读入一个 IN-FILE 的记录数据，就将其中有用的数据送到 DA-FILE 文件的记录区，再输出到 DA-FILE 文件。当然也可以改为用 MOVE CORR 语句实行对应项传送，请读者自己修改此程序。想一想应对哪几处作修改。

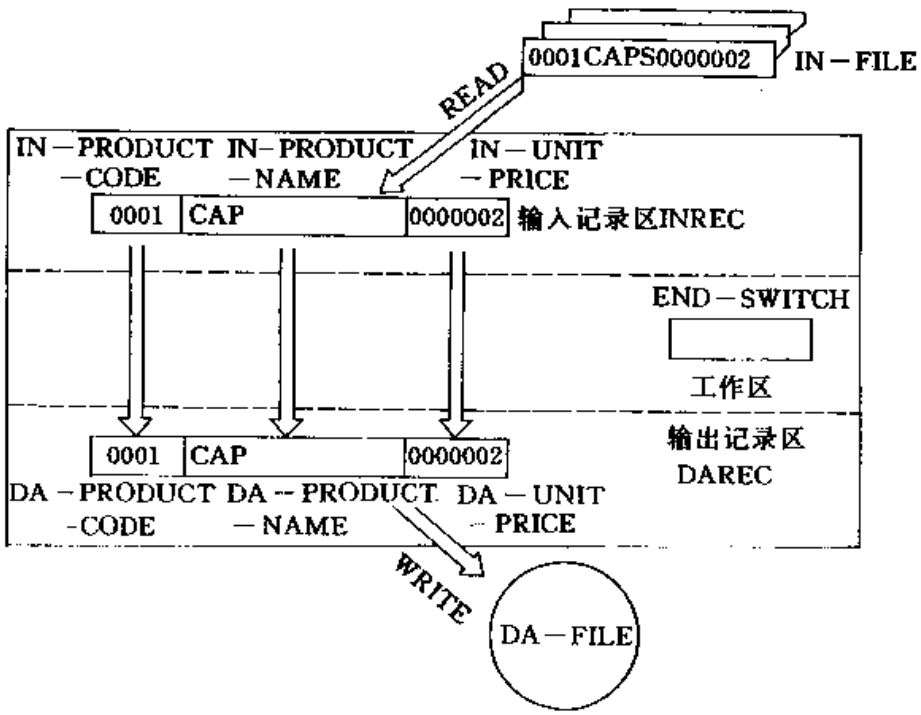



图 10.17

**【例 10.4】** 有一磁盘索引文件，它记录一批产品的以下数据：

| DA-PRODUCT-CODE<br>（产品号） | DA-PRODUCT-NAME<br>（产品名） | DA-UNIT-PRICE<br>（单价） |
|---|---|---|
| X（4） | X（10） | 9（7） |

今用一个“变更文件”（是一个磁盘顺序文件）来修改磁盘索引文件上的数据。“变更文件”的记录有三种不同的格式：

（1）要删除磁盘索引文件上的某一产品的记录（例如该产品已卖完或不再生产了）。

变更文件记录的数据形式为：

| RECORD-TYPE<br>（记录类型） | CD-PRODUCT-CODE<br>（产品号） | BLANK<br>（空格） |
|---|---|---|
| X（2） | X（4） | X（17） |

假如在变更文件记录中的数据为：

DL0010

表示要将磁盘索引文件中产品号为 0010 的记录删除。‘DL’ 是 Delete 的缩写。

（2）增加一个产品记录。

此时变更文件记录中的数据形式为：

| RECORD-TYPE<br>（记录类型） | CD-PRODUCT-CODE<br>（产品号） | CD-PRODUCT-NAME<br>（产品名） | CD-UNIT-PRICE<br>（单价） |
|---|---|---|---|
| X（2） | X（4） | X（10） | 9（7） |

假如变更文件记录中的数据为：

AD0012COAT　　0000025

表示增加一个产品号为0012的新记录到磁盘索引文件中去，显然此时应提供磁盘索引文件记录中所需的全部数据（产品号，产品名，单价）。'AD'是Addition的缩写。

(3) 修改产品的价格。

此时变更文件记录中数据形式为：

| RECORD-TYPE<br>（记录类型） | MT-PRODUCT-CODE<br>（产品号） | MT UNIT-PRICE<br>（单价） | BLANK<br>（空格） |
|---|---|---|---|
| X (2) | X (4) | 9 (7) | X (10) |

如果变更文件记录中的数据为：

UP00140000132

表示将0014号产品的单价改为132。'UP'是Update的缩写。

每次从变更文件读入一条记录（为上述三种形式记录中的任一种），根据读入记录中前两个字符是'DL'，'AD'或'UP'，分别对索引文件进行删除、增加或修改，然后在打印机打印出变更记录的数据和执行的情况，并在每页开始打印表头。格式为：

```
RECORD CONTENT              ERROR/OK MEASSAGE（本行为页表头）
AD0012COAT      0000025     OK STORED（存储完毕）
UP00140000132               OK MODIFIED（修改完毕）
DL0010                      OK DELETED（删除完毕）
UP00800000178               NG UPDATE（没法修改，因无此产品号）
AD0050CELL      0000001     NG OVERFLOW（没有增加进去，溢出，可能由于文件空间已
                            满或索引文件中已有0050产品）
    ⋮
    ⋮
```

打印输出以25行为一页。

程序如下。请读者试着自己按以上的题意看程序，是否能基本上看懂它，然后再看后面的说明和流程图。

```
IDENTIFICATION      DIVISION.
PROGRAM-ID. EXAM 10-1.

ENVIRONMENT         DIVISION.
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
        SELECT  MD-FILE  ASSIGN TO IN-FILE.
        SELECT  DA-FILE  ASSIGN TO INDE
                ORGANIZATION  IS  INDEXED
                ACCESS  MODE IS RANDOM
                RECORD KEY IS DA-PRODUCT-CODE.
        SELECT  LP-FILE  ASSIGN TO P-FILE.
```

```
DATA          DIVISION.
FILE     SECTION.
FD   MD-FILE   LABEL   RECORD IS STANDARD
     DATA RECORD IS MDREC.
01 MDREC.
   02   RECORD-TYPE          PIC X (2).
   02   MD-PRODUCT-CODE      PIC X (4).
   02   CONTENT1             PIC X (17).
   02   CONTENT2   REDEFINES   CONTENT1.
        03   MD-PRODUCT-NAME   PIC X (10).
        03   MD-UNIT-PRICE     PIC 9 (7).
   02   CONTENT3   REDEFINES   CONTENT1.
        03   MD-UNIT-PRICE-R      PIC 9 (7).
        03   FILLER               PIC X (10).
FD   DA-FILE   LABEL   RECORD   IS STANDARD
       DATA   RECORD IS DAREC.
01 DAREC.
   02   DA-PRODUCT-CODE    PIC X (4).
   02   DA-PRODUCT-NAME.
        03 FIRST-BYTE          PIC X.
        03 FILLER              PIC X (9).
   02   DA-UNIT PRICE         PIC 9 (7).
FD     LP-FILE   LABEL   RECORD   IS STANDARD
          DATA   RECORD IS   LPREC.
01   LPREC.
   02   FILLER     PIC X (80).
WORKING-STORAGE SECTION.
77   LINE-COUNT      PIC 99.
77   READ-ERROR      PIC X (3).
77   WRITE-ERROR     PIC X (3).
77   DATA-END        PIC X (3).
77   NOM-KEY         PIC X (4).
01   HEADER.
   02   FILLER     PIC X (11) VALUE   SPACE.
   02   FILLER     PIC X (14) VALUE   'RECORD CONTENT'.
   02   FILLER     PIC X (26) VALUE   SPACE.
   02   FILLER     PIC X (16) VALUE   'ERROR/OK MESSAGE'.
01   DETAILS.
   02   FILLER     PIC X (11)    VALUE SPACE.
   02   RECORD-AREA      PIC X (40).
   02   MESSAGE AREA     PIC X (20).
```

```
PROCEDURE          DIVISION.
MAIN.
      OPEN  INPUT    MD-FILE
            I-O      DA-FILE
            OUTPUT LP-FILE.
      MOVE SPACE TO DATA-END.
      MOVE  25  TO  LINE-COUNT.
   READ  MD-FILE
      AT END MOVE 'END'  TO DATA-END.
   PERFORM  RECORD-PROCESSING
            UNTIL  DATA-END = 'END'.
   CLOSE  MD-FILE DA-FILE, LP-FILE.
   STOP  RUN.
RECORD-PROCESSING.
   MOVE  SPACE  TO READ-ERROR.
   MOVE  MD-PRODUCT-CODE  TO  DA-PRODUCT-CODE.
   READ  DA-FILE
        INVALID  KEY  MOVE  'ERR'  TO  READ-ERROR.
   IF  READ-ERROR -= 'ERR'
      IF  RECORD-TYPE ='AD'
         PERFORM  ADDITION-1
      ELSE
         MOVE 'NG  UPDATE'  TO  MESSAGE-AREA
   ELSE
      IF  FIRST-BYTE = HIGH-VALUE
         IF  RECORD-TYPE = 'AD'
            PERFORM  ADDITION-2
         ELSE
            MOVE  'NG  UPDATE'  TO  MESSAGE-AREA
      ELSE
       IF  RECORD-TYPE = 'UP'
          PERFORM  REVISION
       ELSE
          IF  RECORD-TYPE = 'DL'
             PERFORM  DELETION
          ELSE
             MOVE 'NG  UPDATE'  TO  MESSAGE-AREA.
   RERFORM  PRINT.
   MOVE SPACE TO MDREC.
   READ MD-FILE  AT END  MOVE 'ENDY'  TO  DATA-END.

 ADDITION 1.
   MOVE MD-PRODUCT-CODE  TO  DA-PRODUCT-CODE.
```

```
    MOVE MD-PRODUCT-NAME  TO  DA-PRODUCT-NAME.
    MOVE MD-UNIT-PRICE  TO  DA-UNIT-PRICE.
    MOVE SPACE  TO  WRITE-ERROR.
    WRITE  DAREC  INVALID KEY
           MOVE  'ERR'TO WRITE-ERROR.
    IF WRITE-ERROR = 'ERR'
           MOVE  'NG  OVERFLOW'  TO  MESSAGE AREA
    ELSE
           MOVE  'OK  STORED'      TO  MESSAGE-AREA.

ADDITION-2.
    MOVE  MD-PRODUCT-NAME  TO  DA-PRODUCT-NAME.
    MOVE  MD-UNIT PRICE  TO  DA-UNIT-PRICE.
    REWRITE DAREC INVALID  KEY DISPLAY 'REWRITE ERROR'.
    MOVE 'OK STORED'  TO  MESSAGE-AREA.

REVISION.
    MOVE  MD-UNIT-PRICE-R  TO  DA-UNIT-PRICE.
    REWRITE  DAREC INVALID  KEY DISPLAY 'REWRITE ERROR'.
    MOVE  'OK MODIFIED'  TO  MESSAGE-AREA.

DELETION.
    MOVE  HIGH-VALUE  TO  FIRST-BYTE.
    DELETE DA-FILE INVALID  KEY DISPLAY 'REWRITE ERROR'.
    MOVE  'OK DELETED'  TO  MESSAGE-AREA.

PRINT.
    IF  LINE-COUNT = 25
        MOVE  SPACE  TO  LPREC
        WRITE  LPREC  AFTER  PAGE
        WRITE  LPREC  FROM  HEADER  AFTER  2
        MOVE  0  TO  LINE-COUNT
    ELSE
        NEXT  SENTENCE.
    MOVE  SPACE  TO  RECORD-AREA.
    MOVE  MDREC  TO  RECORD-AREA.
    MOVE  SPACE  TO  LPREC.
    WRITE  LPREC  FROM  DETAILS  AFTER  2.
    ADD  1  TO  LINE-COUNT.
```

本程序中用 MD-FILE 代表“变更文件”(MD 是 modify 的缩写)。MD-FILE 文件为顺序文件组织(省略 ORGANIZATION 子句隐含为顺序文件)。指定 DA-FILE 为索引文件,存取方式为随机存取。指定“记录键”为 DA-PRODUCT-CODE。在环境部文件控制段中,

我们作如下描述：

```
SELECT  DA-FILE  ASSIGN  TO  INDE
    ORGANIZATION  IS  INDEXED
    ACCESS  MODE  IS  RANDOM,
    RECORD  KEY  IS  DA-PRODUCT-CODE.
```

假设我们所用的计算机系统要求在对文件存取数据时按本章 10.5.1 节所介绍的第一种方法给“关键字”项赋以一个值，即直接给 DA-PRODUCT-CODE 赋值。

由于磁盘顺序文件包含三种不同的数据格式的记录，而每个记录长度是相等的。因此在数据部中对“变更文件”用重定义语句以适应三种不同格式的记录。MOREC 的数据组织见图 10.18。

“变更文件”的记录

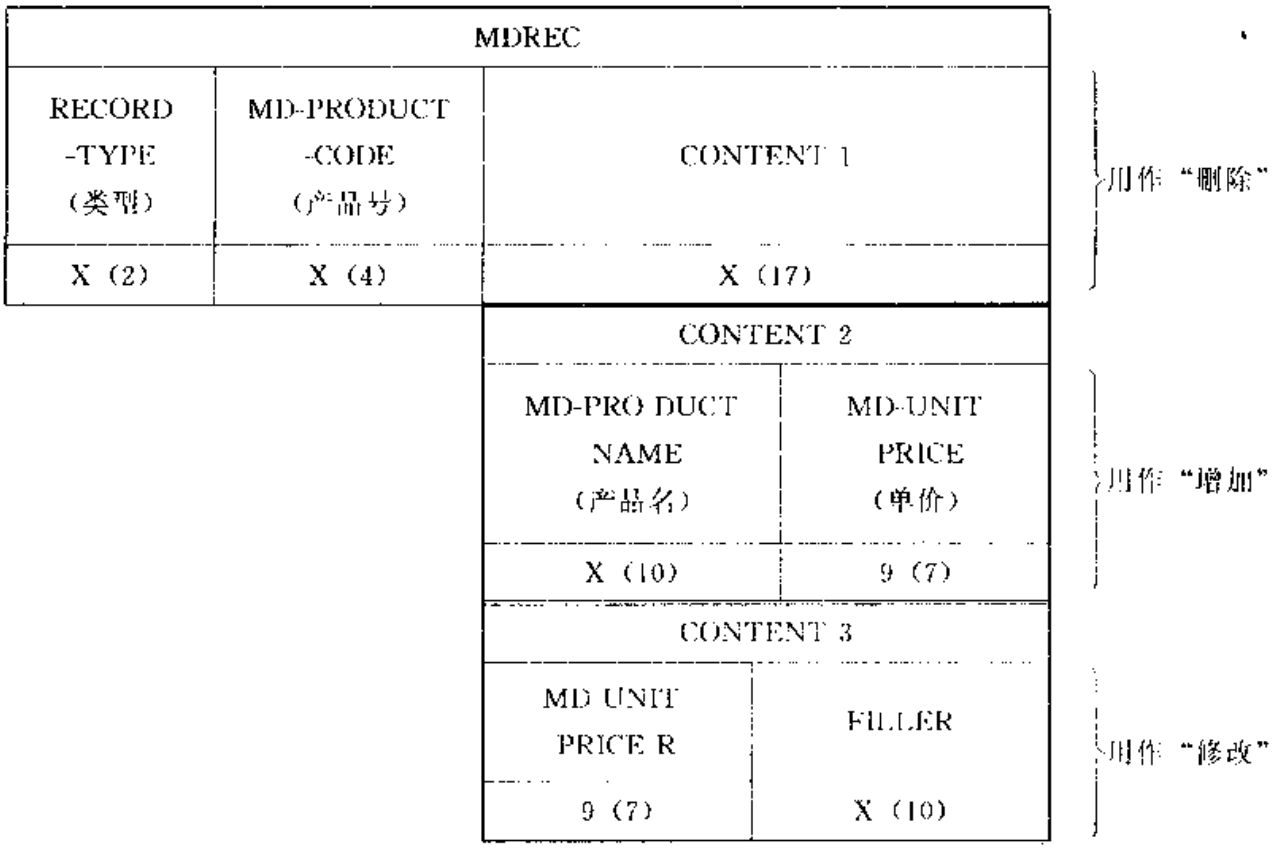



图 10.18

磁盘索引文件记录的数据组织如下：

| DAREC | | | |
|---|---|---|---|
| DA PRODUCT CODE (产品号) | DA-PRODUCT-NAME (产品名) | | DA-UNIT-PRICE (单价) |
| | FIRST BYTE | FILLER | |
| X (4) | X | X (9) | 9 (7) |

上面的 FIRST-BYTE 是 DA-PRODUCT-NAME 项中的第一个字节。稍后我们可以看到 FIRST-BYTE 项是用来存放删除记录的信息的。

在工作单元节中定义的 HEADER（表头）数据结构为：

| HEADER | | | |
|---|---|---|---|
| FILLER | FILLER | FILLER | FILLER |
| X（11） | X（14） | X（26） | X（16） |
| 空格 | RECORD CONTENT | 空格 | ERROR/OK MESSAGE |

项内容

它是在形成打印文件时每页的表头文字。

工作单元节中的数据项 DETAILS 用来指定输出的“细目”，即每页中的各打印行的内容。其格式为：

| DETAILS | | |
|---|---|---|
| FILLER（空格） | RECORD-AREA（记录区） | MESSAGE-AREA（信息区） |
| X（11） | X（40） | X（20） |

每次打印前将“变更文件”中的记录送到 RECORD-AREA，把“是否已正确执行了”的信息（如上面提到的“OK STORED”）传送到 MESSAGE AREA 区。再输出 DETAILS 数据项，则可打印出每一个“变更文件”记录和执行情况。打印文件中一行的输出宽度可根据所使用的打印机情况来设定（今设宽度为 80 列）。

下面我们分析过程部。设计程序时应尽量写成模块式的程序结构。即程序分为若干模块，每一模块（包含若干段）完成某一方面功能。主模块相当于“总调度”，它“组织”各模块来分别完成各工作。这样可以使主模块比较短，程序看起来比较清晰。图 10.19 流程图是对照 MAIN 段主模块画的。它的含义清楚，读一个“变更文件”记录后转去执行“RECORD-PROCESSING”段进行处理。

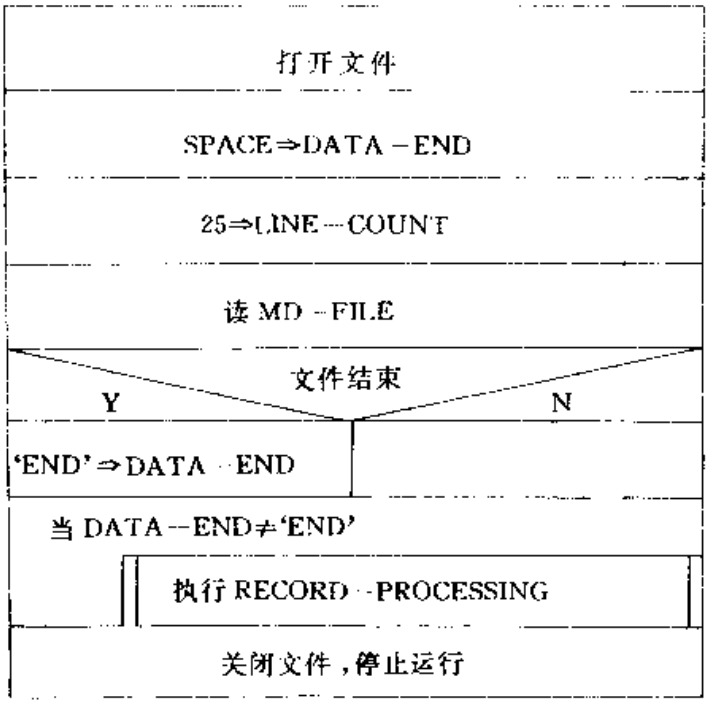



图 10.19

图 10.20 是 RECORD-PROCESSING 段的流程图，为了读一个索引文件记录，应先向“记录键”项传送一个值。MOVE MD-PRODUCT-CODE TO DA-PRODUCT-CODE 的作用是将读入的“变更文件”中记录上的产品号 MD-PRODUCT-CODE 传送给记录键项 DA-PRODUCT-CODE，以便从索引文件中读出含此产品号的记录。下面的 IF 语句，是一个四层的嵌套 IF 语句。请读者对照流程图搞清 IF 语句的逻辑关系。

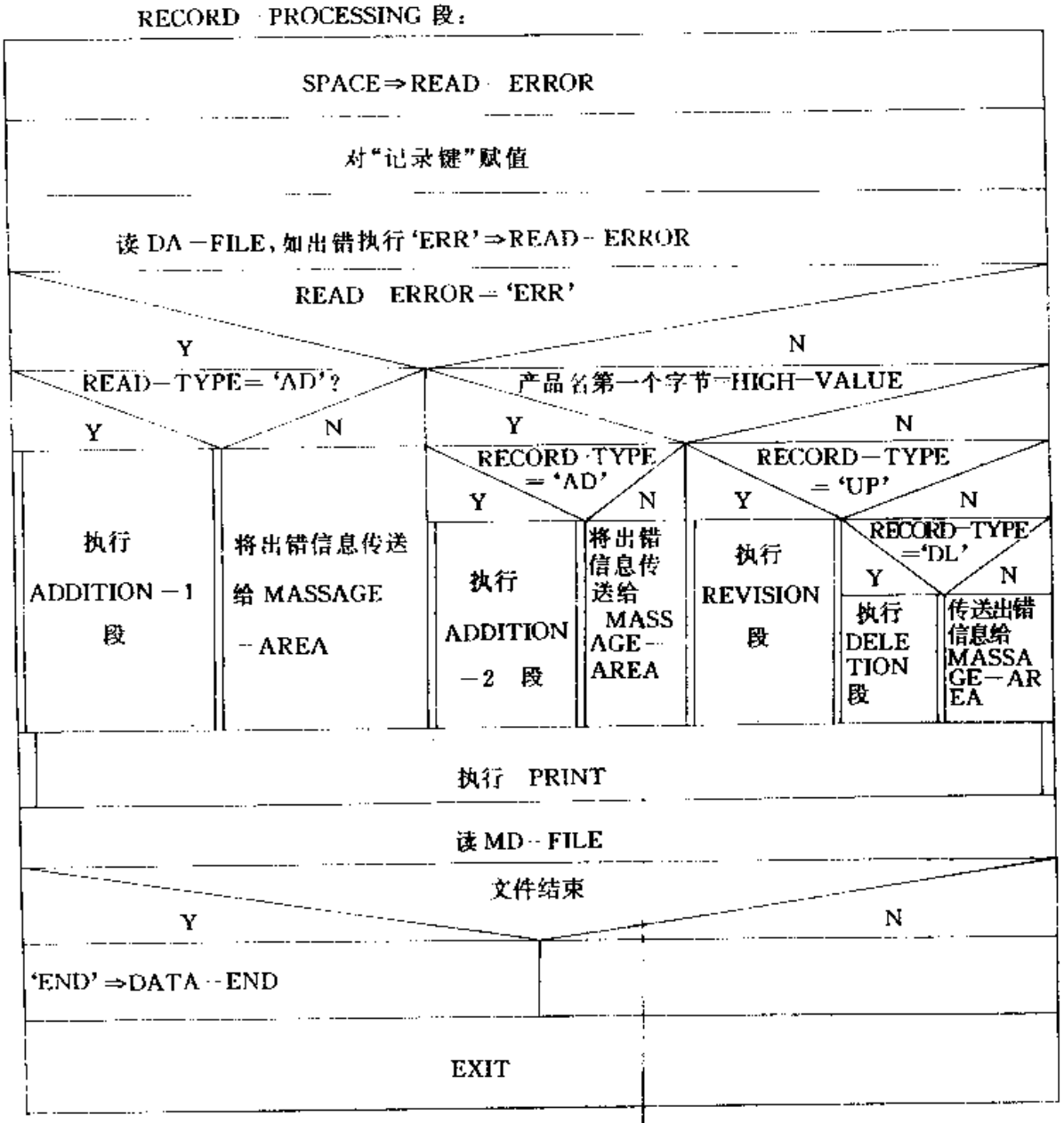



图 10.20

如果输入的“变更文件”记录是“AD0012COAT 0000025”，表示应向索引文件增加一个新记录。我们看在程序中是如何实现的？此时记录键的值为 0012，在读 DA-FILE 时由于在磁盘文件中找不到有些产品号的记录而执行 INVALID KEY 子句，将“ERR”送到 READ-ERROR。IF 语句首先判断 READ-ERROR 的值是否等于“ERR”。如果相等，则表示索引文件上确无此产品号。然后再看 RECORD-TYPE 的值是否“AD”，是的话，则应增加新记录。转去 ADDITION-1 段执行此功能。如果不是“AD”(譬如变更文件的记录为“BD0012COAT 0000025”，显然是弄错了(把“AD”错成“BD”)，则不应执

行“增加新记录”的功能。而按“出错”处理。图 10.21 为 ADDITION-1 段的流程图。

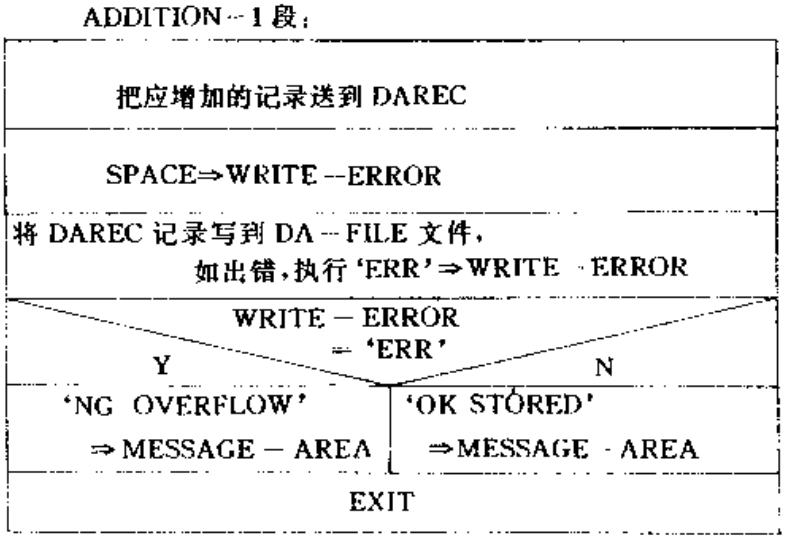



图 10.21

如果输入的变更记录为“DL0001”，表示要删除索引文件中产品号为 0001 的记录。IF 语句判断 READ-ERROR 值不是“ERR”，则应执行 ELSE 子句，进入第二层 IF ELSE 语句。由于产品号的第一个字节（即 FIRST-BYTE）的内容不是 HIGH-VALUE（二进位全“1”），则应执行第二层的 ELSE 子句，并进入第三层的 IF-ELSE 语句。IF 语句判断 RECORD-TYPE 不等于 'UP' 则执行第三层的 ELSE 子句，并进入第四层 IF-ELSE 子句。由于 RECORD-TYPE 值等于“DL”，故执行“DELETION”段，进行“删除”记录。见图 10.22。

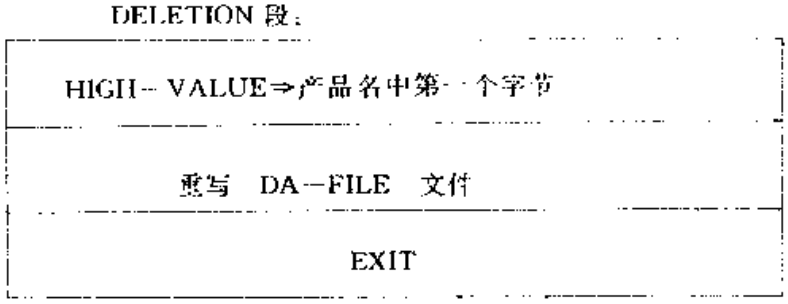



图 10.22

事实上并不是在物理上将该记录从磁盘上抹掉，而只是将 PRODUCT-NAME 项中第一个字符（即 FIRST-BYTE 项）的值变为 HIGH-VALUE（即二进制的全“1”）。以后再根据此项是否 HIGH-VALUE，决定如何处理。如果想删除一个记录，但该记录中的 FIRST-BYTE 值已是 HIGH-VALUE，则表示已被删除了，不能再删除。如 RECORD-TYPE 项是“AD”，则应增加新记录。由于磁盘索引文件的记录中该产品名的第一个字符已为 HIGH-VALUE，说明是先前被“删除”的。因此，此时的“增加”不是从物理上真正再加入一个新记录，而是将原记录“恢复”。执行“ADDITION-2”段（图 10.23）。将变更文件上的产品名 MA-PRODUCT-NAME 再送到磁盘索引文件记录中的 DA-PRODUCT-NAME 中，将 FIRST-BYTE 项的 HIGH-VALUE 冲掉，这样此记录又恢复原状，成为有效。如果单价改变了，则应将 MA-UNIT-PRICE 送到 DA-UNIT-PRICE 中。

如果变更记录为“UP00140000132”，表示要修改产品号为 0014 的磁盘索引文件记录。如果读出的含此产品号的记录的 FIRST-BYTE 项值为 HIGH-VALUE，则按出错处理。如不是 HIGH-VALUE，记录的前两个字符（RECORD-TYPE 项）为“UP”，则执行“REVISION”段实现修改（图 10.24）。如果此时前两个字符既非“UP”，又非“DL”，则不修改，不删除，显然应按出错处理。

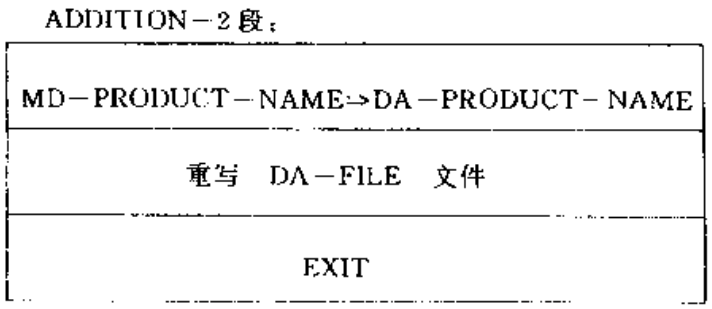



图 10.23

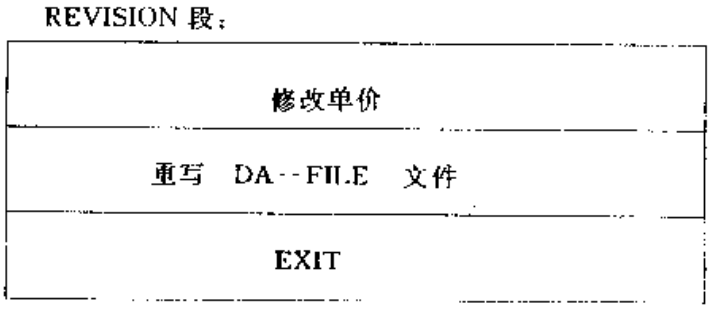



图 10.24

在对磁盘索引文件记录进行处理以后，打印出本次变更记录内容和如何进行处理的信息。这是由“PRINT”段实现的。要求打印出信息的格式已在题目中列出。流程见图 10.25。这部分读者是容易看懂的。

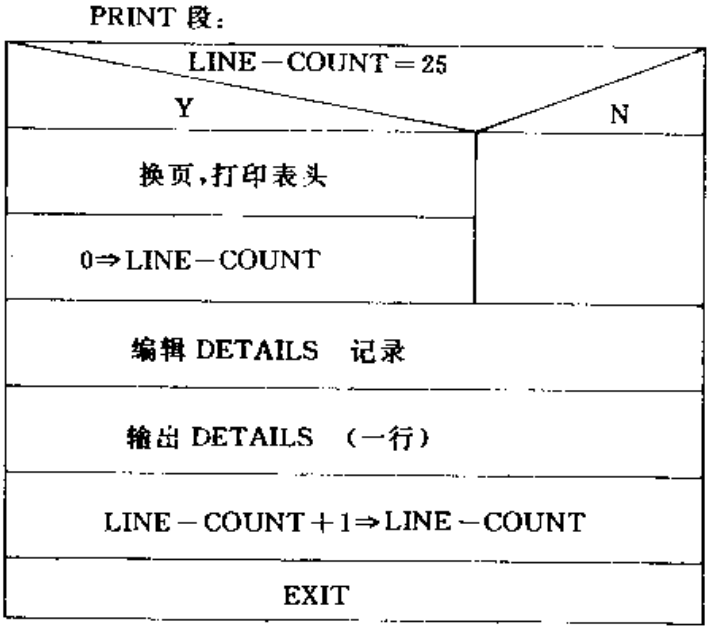



图 10.25

## §10.6 磁盘相对文件

### 10.6.1 相对文件的概念

所谓相对文件，就是在建立文件时，除了记载记录本身之外，还给每一记录编一个“位置号”。以后就按指定的记录位置号存取记录。如果是用顺序写方式向相对文件写记录，则“记录位置号”是按记录存入的顺序确定的，例如，最先存入的记录为1号记录，第二个存入的为2号记录，等等。

和顺序文件不同，相对文件中允许有空记录。什么位置出现空记录由程度设计者安排。

下面是一个相对文件的示意图：

| 记录位置号 | 1# | 2# | 3# | …… | K# | …… | N# |
|---|---|---|---|---|---|---|---|
| 记　　录 | $REC_1$ | 空 | $REC_2$ | | 空 | | $REC_n$ |

要定义一个“相对键”(RELATIVE KEY)，如果已使它的值为3，则在执行READ语句时，就直接读出第三个记录。

相对文件的存取比较简单，使用也较方便。

### 10.6.2 COBOL中与相对文件有关的成分

·在环境部中的SELECT子句里要对相对文件作说明。

```
SELECT 文件名ASSIGN TO 设备名
  ORGANIZATION IS RELATIVE
  ACCESS MODE IS {SEQUENTIAL [, RELATIVE KEY IS 数据名1]
                 {RANDOM, RELATIVE KEY IS 数据名2
```

·过程部的OPEN（三种打开方式），CLOSE，READ，WRITE，REWRITE，START，DELETE等语句的含义与前述相同，不再重复。

说明：

（1）相对文件可以顺序读（应说明ACCESS MODE IS SEQUENTIAL），此时可以不指定RELATIVE KEY。可以用START语句指定顺序读的起点（如不用START指定，则从第一个记录开始顺序读）但用START语句时，应写RELATIVE KEY子句。因为START语句中的KEY是要和RELATIVE KEY的值进行比较的。如果顺序读遇到空记录或被删除的记录，则跳过，直到读出下一个实记录为止。

（2）作为RELATIVE KEY（相对键）的数据项不能是相对文件的记录中的一项。这是与索引文件不同的（索引文件的记录键RECORD KEY项必须是记录中的一项，按它的值的递增顺序将记录排列），或者说，相对文件的“记录位置号”是记录外的一项，并不是加到记录里面的。

（3）随机读时应事先给“相对键”项送入一个值（记录位置号），如果这个值超过文件

最大记录号(即要求读文件中不存在的记录)或此记录是一个空记录或已被删除的记录,则执行 READ 语句中的 INVALID KEY 子句中的强制语句。

(4)WRITE 语句是将一个新记录写入文件,但只准在空记录位置上写,否则就执行 INVALID KEY 子句中的强制语句。

(5) 用 WRITE 语句建立一个新的相对文件时,既可以顺序写。也可以随机写。用顺序方式建立时,记录一个一个写入文件,中间不能留空记录。

随机写,是按"相对键"数据项的值直接指向该"记录位置号",在此位置上写入一个新记录。如果我们在 1,3,5,7……位置上写上了记录,而 2,4,6,8,……位置上未写,则 2,4,6,8…位置便是空记录。

(6) REWRITE 语句用来更新一个记录,而 WRITE 语句只能往空记录上写。如果指定的是顺序存取方式 (ACCESS MODE IS SEQUENTIAL),则 REWRITE 是写回刚读出记录的位置上,相对键此时不起作用 (但用 START 时例外)。

用随机存取方式时,不必先读入 (READ) 记录,而可按指定相对键的值用 REWRITE 直接更新文件中原记录。当 RELATIVE KEY 指出的记录位置号在文件中不存在时,执行 INVALID KEY 子句中的强制语句。

(7) DELETE 语句从文件中删除一个实记录。如果是顺序存取方式,则删除刚读出的记录。如果是随机存取方式,则根据 RELATIVE KEY 数据项的值指出的记录位置,删除该记录。如果不存在该位置或该位置上为空记录,则执行 INVALID KEY 中的强制语句。

### 10.6.3 相对文件应用举例

**【例 10.5】** 有一个磁盘顺序文件,包含 50 个记录,每个记录包括学生号、学生名、学生成绩。要求先将这些记录读入内存,然后建立一个相对文件,将此 50 个记录写到第 1,3,5,7……位置上,将 2,4,6…位置留空,以便以后插入新的记录。

流程图见图 10.26

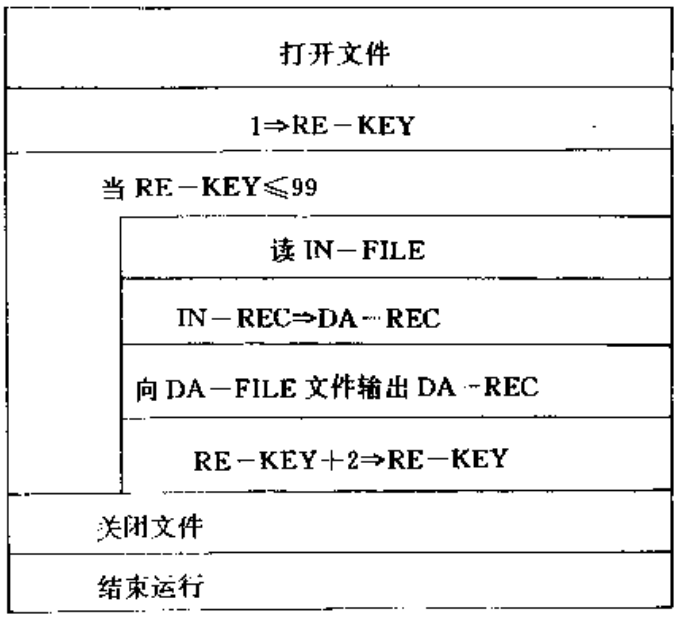



图 10.26

程序如下：

```
IDENTIFICATION   DIVISION.
PROGRAM-ID. EXAM10-5.
ENVIRONMENT      DIVISION.
INPUT-OUTPUT SECTION.
FILE CONTROL.
        SELECT IN-FILE ASSIGN IN-FILE.
        SELECT DA-FILE ASSIGN REL-FILE.
           ORGANIZATION IS RELATIVE
           ACCESS MODE IS RANDOM
           RELATIVE KEY IS RE-KEY.
DATA        DIVISION.
FILE     SECTION.
FD   IN-FILE LABEL RECORD STANDARD.
01   IN-REC.
     02 STUDENT-NUM PIC 9(6).
     02 STUDENT-NAME PIC X(20).
     02 STUDENT-SCORE PIC 999.
FD DA-FILE LABEL RECORD STANDARD.
01   DA-REC.
     02 STUDENT-NUM PIC 9(6).
     02 STUDENT-NAME PIC X(20).
     02 STUDENT-SCORE PIC 999.

WORKING-STORAGE SECTION.
77   RE-KEY      PIC 99.

PROCEDURE        DIVISION.
STAR.
      OPEN INPUT IN-FILE   OUTPUT DA-FILE.
EXC.
      PERFORM RECORD-PROCESSING
                  VARYING RE-KEY FROM 1 BY 2
                     UNTIL RE-KEY > 99.
TERM.
      CLOSE IN-FILE DA-FILE.
      STOP RUN.

RECORD-PROCESSING.
      READ IN-FILE AT END GO TO TERM.
      MOVE CORR IN-REC TO DA-REC.
      WRITE DA-REC
            INVALID KEY DISPLAY 'ERROR', RE-KEY, DA-REC.
```

说明："相对键"项指定为RE-KEY。在开始执行RECORD-PROCESSING段时，RE-KEY的值为1，在读入一个记录并将有用数据传送到DA-REC后，将DA-REC写到记录位置号为1的位置。然后RE-KEY增值为3，再读入下一个记录，将其数据写到位置号为3的记录位置上，……直到将第50个记录数据写到第99号记录，然后RE-KEY增值为101，不再执行RECORD-PROCESSING，结束运行。

## §10.7 动态存取方式简介

索引文件和相对文件既可以随机存取，又可以顺序存取。ANSI COBOL 1974还包括了一种动态(DYNAMIC)存取方式。所谓动态存取方式就是指在同一程序中既可用顺序方式存取，又可用随机方式存取，由所用的输入语句决定。因此，严格地说，动态存取方式不是一种独立的存取方式，而只是顺序存取方式和随机存取方式的结合，由程序选择其中之一罢了。

如果用动态存取方式，应在设备部的SELECT子句中将ACCESS子句改写为：

```
ACCESS  MODE  IS  DYNAMIC
```

这样在程序中既可随机存取，又可顺序存取。那末究竟根据什么来决定具体执行哪一种方式的存取呢？COBOL为此增加了一个语句：

| READ 文件名 NEXT RECORD [INTO 标识符] [; AT END 强制语句] |
|---|

例如：READ DAFILE NEXT AT END GO TO TERM.

这个READ语句比原来的READ语句多一个"NEXT"，它的意思是"读下一个记录"，也就是"顺序读"的意思。在DYNAMIC方式中，如只写READ，不写NEXT，则表示按随机方式读。那么顺序读时"下一个"记录从哪里开始算呢？以当前记录为起点。指针由系统设定，用户不必定义它。如果刚才用READ语句（随机读）读出第14号记录，现在用READ NEXT语句就能读出第15号记录。但用READ NEXT时，如遇到相对文件中的空记录，跳过此记录，取下一个实记录。

最好是用START语句来确定READ NEXT的读操作的初始记录号。假如要读一相对文件（其"相对键"已指定为RE-KEY），可以写成：

```
MOVE 10 TO RE-KEY.
START DAFILE KEY > RE-KEY
      INVALID KEY GO TO ERR.
READ DAFILE NEXT RECORD
      AT END GO TO TERM.
  ⋮
```

今RE-KEY的值为10，START语句规定起始点应大于RE-KEY，即应大于10，则执行READ NEXT语句时读出第11号记录。如果11号为空记录，则读第12号记录，……。如果START语句改为：

```
START DAFILE KEY = 10
INVALID KEY GO TO TERM.
```

则执行 READ NEXT 语句时读第 10 号记录。

请注意：(1) READ 语句有两种形式（用 READ…NEXT 形式的语句顺序读；用不带 NEXT 的 READ 语句随机读），而对于 WRITE 和 REWRITE，DELETE 语句来说，没有 NEXT 可选项，只能用随机方式处理，不论指定的是随机存取方式或动态存取方式，都按 RELATIVE KEY 数据项指定的记录号写出或删除。

(2) 对索引文件的 READ NEXT 语句的用法也是一样的。也先用 START 语句定位。

下面列出文件（索引文件或相对文件）的打开方式和输入/输出语句之间的关系。

| 存取方式 | 语句 | OPEN 方式 | | |
|---|---|---|---|---|
| | | INPUT | OUTPUT | I-O |
| 顺序存取 SEQUENTIAL | READ | √ | | √ |
| | WRITE | | √ | |
| | REWRITE | | | √ |
| | START | √ | | √ |
| | DELETE | | | √ |
| 随机存取 RANDOM | READ | √ | | √ |
| | WRITE | | √ | √ |
| | REWRITE | | | √ |
| | START | | | |
| | DELETE | | | √ |
| 动态存取 DYNAMIC | READ | √ | | √ |
| | READ NEXT | √ | | √ |
| | WRITE | | √ | √ |
| | REWRITE | | | √ |
| | START | √ | | √ |
| | DELETE | | | √ |

## 习　题

10.1　有几种文件的组织形式？请写出它们的名称和记录在文件中排列的原则。

10.2　有几种存取方法？请写出它们的名称及其特点。

10.3　磁带文件、磁盘文件、打印文件各是什么文件组织形式？对它们应该用什么方式存取？

10.4　索引文件的索引项是用什么键（KEY）指定的？怎样从一个索引文件中读出所需的记录？

10.5　怎样从相对文件中读出所需的记录？

10.6　修改补充例 10.1 程序，在建立好该磁带文件后，再逐个读出磁带文件中各记录，并打印之。

10.7　假设上题的磁带文件中有若干个记录，今要求将该文件中第 10，20，30…等逢十的记录取消（删去），请写一程序。（提示：需另建立一个新磁带文件，将老磁带文件中除第 10，20，30…记录外的全部记录写到这个新的磁带文件上。）

10.8 已建立了一个顺序磁盘文件，内有一个年级的若干学生的分数记录，格式为：

| 学　　号 | 姓　　名 | 分　　数 |
|---|---|---|
| 9（6） | X（20） | 9（3）V99 |

今有一个“变更文件”，用来修改学生成绩，其记录格式为：

| 学　　号 | 分　　数 |
|---|---|
| 9（6） | 9(3) V99 |

请编写程序，按输入记录上指定的学号和成绩修改磁盘文件上相应的记录（修改后的成绩送回磁盘文件）。

将每一学生的成绩（修改后的和不需修改的）都打印出来（包括学号、姓名、成绩）。如果该学生的成绩是修改过的，在备注一栏中打印出“UPDATED”。如果在磁盘文件中找不到卡片要求修改的学生记录（无此学号），则打印出该学号并在备注栏中打印“HAS NOT BEEN FOUND”。

打印记录格式为：

| 学　号 | 空　格 | 姓　名 | 空　格 | 成　绩 | 空　格 | 备　注 |
|---|---|---|---|---|---|---|
| 9（6） | X（10） | X（20） | X（10） | Z（3）.ZZ | X（10） | X（84） |

10.9 已有一磁盘文件 FILE1，其格式为：

| CD-PRODUCT-CODE（产品号） | CD-PRODUCT-QTY（产品数量） |
|---|---|
| 9（4） | 9（5） |

另有一磁盘文件 FILE2，格式为：

| DA-PRODUCT-CODE（产品号） | DA-PRODUCT-NAME（产品名） | DA-UNIT-PRICE（单价） |
|---|---|---|
| 9（4） | X（20） | 9（7） |

此磁盘文件是索引文件，RECORD KEY 是 DA-PRODUCT-CODE。

要求从磁盘文件 FILE1 读入数据，按该产品号从磁盘索引文件 FILE2 中找出相应的记录，将 FILE1 文件上提供的产品数量与产品单价相乘，得出总价，将计算结果打印出来，每一产品打印一行。

# 第十一章　排序与合并

## §11.1　排序的概念

在数据处理中，常常要求对一批数据按大小排列次序。例如，一个部门所属500个工厂，每日上报产量和产值，要求迅速地按产值高低将500个工厂排队。又如普查人口，要求将全国各县按人口多少排次序。学校每学期希望列出全校考试成绩最好的100名学生。县政府可能希望按亩产量的高低对各村排队，……等等。排序是数据处理的最重要、最基本方法之一，不少经济活动的数据，只有按指定的顺序排列才能为经济分析提供有力的参考资料。据统计，在数据处理中，排序分类的工作约占全部工作量的30～70%。COBOL具有排序的功能，这就大大减轻了人工劳动，并有可能提供过去人工进行数据处理所不能提供的资料，这也是COBOL比其它高级语言（例如FORTRAN，PASCAL…）更适合用于数据处理的重要原因之一。

假如有一批学生的记录，每个记录包括：姓名、年龄、年级。原来是无规律地排列的：

| 姓　　名 | 年龄 | 年级 |
|---|---|---|
| ZHANG LI | 18 | 2 |
| WANG HAO | 19 | 1 |
| LI FUN | 18 | 1 |
| FUNG MIN | 20 | 4 |
| LING WU | 20 | 3 |

如果要求以姓名的字母次序排列，其次序应为：

| | | |
|---|---|---|
| FUNG MIN | 20 | 4 |
| LI FUN | 18 | 1 |
| LING WU | 20 | 3 |
| WANG HAO | 19 | 1 |
| ZHANG LI | 18 | 2 |

如果要求按年龄由小到大排列，若年龄相同，再按年级由高到低排列。则其次序为：

| | | |
|---|---|---|
| ZHANG LI | 18 | 2 |
| LI FUN | 18 | 1 |
| WANG HAO | 19 | 1 |
| FUNG MIN | 20 | 4 |
| LING WU | 20 | 3 |

当然还可以要求以年级由高到低排列等等。因此，可以看出，如果一个数据记录有几个数据项的话，应该指定一个作为“排序项”，即以此项的值来作为记录排序的依据。有时，可以有两个“排序项”，如上面第二次排序，以年龄为主排序项，如果年龄相同，再以年级高低排序，“年级”就是“副排序项”。在COBOL中称排序项为“排序键”。所谓“键”，指的是“关键字”（KEY WORD），即按哪一个关键字（即数据项）作为排序的依据。

除了指定“排序键”之外，还要指出是按“升序”还是按“降序”排列。譬如，年龄由小到大排序就是升序，年级由高到低排列就是降序。字符的大小比较以什么为标准呢?数字以0⇒9为升序，字母以A⇒Z为升序，即Z“大于”A。各个计算机对其所用的字符规定了大小的顺序。一般采用以下两种方法之一：(1) 按ASCII代码比较，在ASCII码中，0为八进制的060，A为八进制的101，因此，A>0，空格是040，比数字和字母都小，(2) 按EBCDIC代码，0是十六进制的“F0”，A是“C1”，因此0>A。

所以，在排序前应先弄清楚所用的计算机COBOL采用的代码。有的计算机（如PDP 11，IBM-PC）是用ASCII码。有的计算机（M系列，IBM中型机）用EBDIC码；但共同的规律是：A<Z，0<9，至于字母、数字与其它专用字符（如等号=，分号；等）之间相比谁大谁小，需要查所用计算机的规定。

## §11.2 实现排序的步骤

一个需要排序的文件存储在外部介质上（如磁盘）。为了实现排序，还必须提供临时的中间工作文件，并在内存相应地开辟临时的工作记录区，以便存放参加排序的记录，PDP-11要求至少定义三个磁盘文件作中间文件用。M150F要求提供3～9个磁带文件或1～8个磁盘文件作为中间工作文件，而VAX系列机只需定义一个磁盘文件。至于怎样具体地利用中间文件进行排序，有各种不同的实现方法，用户可不必详细了解。

要进行排序，应排行以下三个步骤：

（一）建立排序中间工作文件。从输入文件中依次输入待排序的各记录，送到中间文件的记录区中，再送到磁盘（带）上，建立中间文件。中间文件是建立在磁盘或磁带（可重用介质）上的顺序文件。反复执行图11.1中①、②、③三步，每次输入一个记录。最后，输入文件的记录已全部转写到中间文件中去了，输入文件的任务已经完成，以后不必再用到它了。

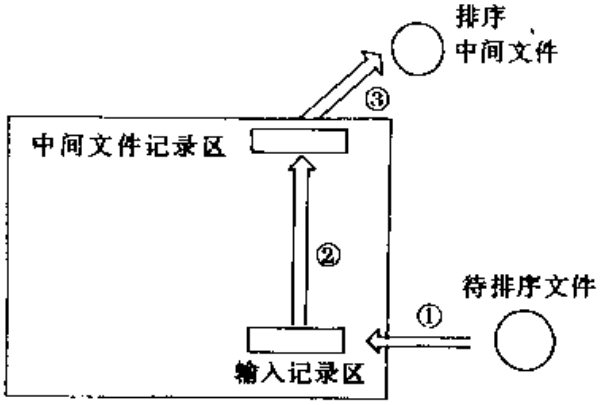



图 11.1

（二）对排序中间文件中的各记录，按指定的排序键和升降序要求进行排序。排序是由计算机自动进行的，用户不必过问它是如何进行的。应该注意的是，排序的对象是**排序中间文件**而不再是输入文件了。经过排好序的记录仍存放在排序中间文件中，但此时记录已按要求排好序了。

（三）排好序的中间文件中的各记录，可以输入到内存的记录区供使用，也可以输出到

另一介质上建立一个新的顺序文件。有人可能会问，在中间文件上不是已有排好序的文件了吗？问题不已解决了吗？为什么还要另建立一个新文件来存放排好序的记录呢？这是因为中间文件是程序为排序而临时设立的文件，当程序运行结束后，它就撤消了。因此必须在程序结束运行之前将它转送到另一个文件中保存。

以上步骤可以用图11.2简单示意。

归纳起来，为实现排序，需要**三个步骤、三个文件**。

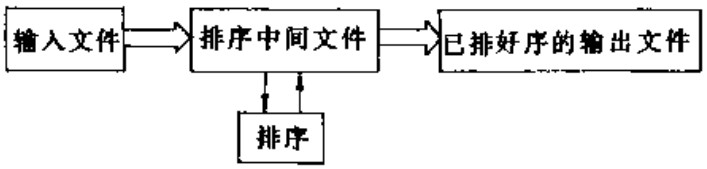



图 11.2

## §11.3 COBOL 中与排序有关的成分

（一）由于排序需要三个文件，因此需要在环境部分别为这三个文件分配设备。除了对输入文件和输出文件需要如前述的分配设备外，还需要对排序中间工作文件分配设备。如：

SELECT SORT-WORK-FILE ASSIGN TO 磁盘文件。

其中 SORT-WORK-FILE 为程序设计者指定的排序中间工作文件，名字是任意起的。用 SORT-WORK-FILE 作名字是为了从文件名中就可以直接看出这个文件是排序工作文件，便于理解程序。

文件组织方式应指定为顺序的，存取方式也是顺序的（在一般计算机系统的 COBOL 中规定，如果不写 ORGANIZITION 和 ACCESS MODE 子句，则隐含表示是顺序文件，存取方式也是顺序的）。

输出文件也要指定设备。如：

SELECT SORTED-FILE ASSIGN TO 磁盘文件。

用 SORTED-FILE 作输出文件名，表示“已排好序的文件”，也是为了一目了然。

（二）在数据部中，要对每一文件进行数据描述。输入和输出文件的数据描述已如前述。排序中间工作文件的描述体以 SD 开头，即以 SD 为层指示符（或称文件描述符），而不是以 FD 为层指示符。如：

SD SORT-WORK-FILE DATA RECORD IS SORT-REC.

以前用的 FD 是 File Description（文件描述）的缩写，而 SD 是 Sort-file Description（排序文件描述）的缩写。了解这一点后就不必死记，自然就会记住了。

对排序中间文件不必指定 LABEL RECORD IS STANDARD，也不能组块，不能写 BLOCK CONTAINS 子句。它的一般格式为：

SD 中间文件名 [RECORD CONTAINS [整数1 TO] 整数2 CHARACTERS]
[DATA {RECORD IS / RECORDS ARE} 数据名1 [，数据名2] …]

对排序中间文件记录中各数据项的描述一般应当和输入文件记录中各数据项的描述相同。因为输入文件记录中各数据项的值要传送到排序中间文件去，并要按排序中间文件记录区中的排序键进行排序，如果二者描述不同，就会出现错误。例如输入文件的描述为：

```
FD  INFILE LABEL RECORDS ARE OMITTED.
01  INREC.
  02  A  PIC  X(10).
  02  B  PIC  9(8).
  02  C  PIC  X(26).
  02  D  PIC  X(36).
```

排序中间文件的描述应为：

```
SD SORT WORK-FILE.
01 SORTREC.
  02 A1 PIC X(10).
  02 B1 PIC 9(8).
  02 C1 PIC X(26).
  02 D1 PIC X(36).
```

排序中间工作文件的记录 SORTREC 和待排序文件的记录 INREC 中各数据项所用的名字不同（如 A1和 A，B1和 B），但数据描述是相同的（例如 A 和 A1都用 PIC X（10）描述）。

如果写成：

```
01 SORTREC.
  02 A1 PIC X(8).
  02 B1 PIC 9(10).
  02 C1 PIC X(30).
  02 D1 PIC X(32).
```

就会引起混乱，排序的结果往往非所预料。

（三）在过程部中用来排序的语句，主要的就是一个 SORT 语句，它是一个功能很强的排序语句，我们将在下节详述。另外有两个用于排序的辅助语句：

RELEASE（释放）语句和 RETURN（回收）语句，也在后面介绍。

## §11.4 SORT 语句的第一种形式

如果一个文件只需要进行排序，则可用这个方法。下面通过一个例子来说明。

**【例11.1】** 已有一数据文件，记录了顾客买货的数据，其格式为：

| YY-MM-DD | CUSTOMER-NUM (顾客号) | CUSTOMER-NAME (顾客名) | PRODUCT-CODE (货号) | PRODUCT-NAME (货名) | QTY (数量) | AMOUNT (金额) |
|---|---|---|---|---|---|---|
| 9 (6) | 9 (8) | X (10) | X (6) | X (10) | 9 (6) | 9 (8) V99 |

要求按顾客号升序排序。如果顾客号相同，再按日期升序排序，如果日期又相同，按金额降序排列，将排好序的记录输出到磁盘上。

先分析题目：1. 需要指定排序键。现在需要指定主排序键为顾客号 CUSTOMER-NUM，升序。副排序键有两个：日期（YY-MM-DD），升序；金额（AMOUNT），降序。

2. 设三个文件：

（1）待排序文件为磁盘数据文件，定名为 IN-FILE。

（2）排序中间工作文件，定名为 SORT-WORK FILE，建立在磁盘上。

（3）已排好序的输出文件，定名为 SORTED-FILE，建立在磁盘上。

程序如下：

```
IDENTIFICATION      DIVISION.
PROGRAM-ID. EXAM11-1.
ENVIRONMENT         DIVISION.
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
       SELECT  IN-FILE    ASSIGN  TO  IN-FILE.
       SELECT  SORT-WORK-FILE    ASSIGN  TO  SW.
       SELECT  SORTED FILE    ASSIGN  TO  SORT-FILE.
DATA        DIVISION.
FILE    SECTION.
FD    IN-FILE  LABEL  RECORD  IS  STANDARD  (输入文件)
      DATA  RECORD  IS  INREC.
01  INREC.
    02  YY-MM-DD           PIC 9(6).
    02  CUSTOMER-NUM       PIC 9(8).
    02  CUSTOMER-NAME      PIC X(10).
    02  PRODUCT CODE       PIC X(6).
    02  PRODUCT NAME       PIC X(10).
    02  QTY                PIC 9(6).
    02  AMOUNT             PIC 9(8)V99.
SD    SORT-WORK-FILE.
01  WORKREC.
    02  YY-MM DD-S         PIC 9(6).
    02  CUSTOMER-NUM-S     PIC 9(8).
    02  CUSTOMER NAME-S    PIC X(10).
    02  PRODUCT-CODE-S     PIC X(6).
    02  PRODUCT-NAME-S     PIC X(10).
    02  QTY-S              PIC 9(6).
    02  AMOUNT-S           PIC 9(8)V99.
FD    SORTED FILE  LABEL  RECORD  IS  STANDARD.(已排序文件)
01  SORTEDREC          PIC  X(80).
PROCEDURE           DIVISION.
SORTING.
```

```
SORT   SORT-WORK-FILE
  ON   ASCENDING KEY CUSTOMER-NUM S
                     YY-MM-DD-S        (升序键)
  ON   DESCENDING KEY AMOUNT-S         (降序键)
  USING   IN-FILE
  GIVING   SORTED FILE.
  STOP   RUN.
```

说明：

1. 读者可以看到输入文件和中间文件的记录描述格式是相同的，只是所用的名字不同。输出文件的记录用 PIC X (80) 描述，即把排好序的记录中的数据全部按字符输出到磁盘文件中去。如果在本程序中不再使用它的话，是可以这样描述的，而不必再分为若干初等项并逐项描述。但是如果下次需要调出来用时，则应在数据部中对其记录中的每一项分别描述，以便能分别使用其中的每一项的数据。

2. 过程部的 SORT 语句用来实现排序的三个步骤。它的作用是：

(1) 指出本语句是实现排序的语句。

(2) 指出排序中间文件名字，进行排序时是以中间文件为对象进行排序的。

(3) 指出排序键（主键和副键）、升序或降序。ASCENDING 表示升序，DESENDING 表示降序。

(4) 指出中间文件的记录是从哪个输入文件送来的，用 USING 短语指定。

(5) 指出将排好序的中间文件的记录送到哪一个文件中去，用 GIVING 短语指定。

可以看出这一个 SORT 语句功能很强，实现了多方面的操作。

本程序中的 SORT 语句的作用是：①由输入文件 IN FILE 将各记录传送给排序中间工作文件。②再对这个中间文件 SORT-WORK FILE 实行排序。主排序键是 CUSTOMER-NUMS，升序。第一副排序键是 YY MM-DD-S，升序。第二副排序键是 AMOUNT-S，降序。③排好序后的记录送到 SORTED-FILE（输出文件）中去。

3. USING 的作用是自动将输入文件的各记录依次送入内存，然后传送到排序中间文件的记录区，再写到中间文件中去。它自动实现了图11.1中所表示的①、②、③三个步骤。

注意，不必在过程部中打开和关闭输入文件。它们是自动实现的。

4. GIVING 的作用是将已排好序的中间文件的记录逐个地调入内存，然后传送给输出文件的记录区，再输出到输出文件 SORTED-FILE 上去。即图11.3所示的①、②、③三步。

同样，对输出文件的打开和关闭也是自动完成的。不必在程序中写 OPEN 和

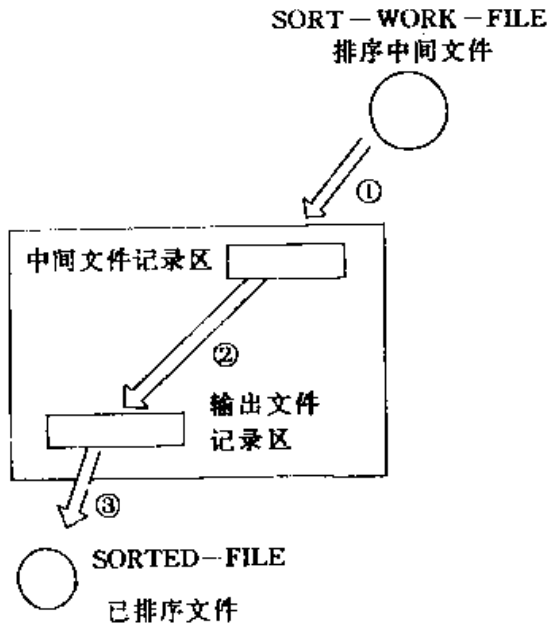



图 11.3

CLOSE 语句。

5. 注意 SORT 后面的文件名是中间文件的名字 SORT-WORK-FILE，而不是误写为输入文件的名字 IN-FILE。这是不少初学者常常易搞错的。从输入文件将各记录传送给中间文件后，输入文件的作用已经完成了。排序是对中间工作文件进行的，而不是对原来的输入文件进行排序。排好序的结果仍放在中间文件中。因此，排序中间文件在排序前和排序后记录排列的顺序是不同的。

6. 作为排序键的数据项不能含 OCCURS 项。也不能从属于含有 OCCURS 子句的数据项。如：

```
01  A.
    02  B OCCURS 10 PIC X(10).
```

B 是不能作排序键的。排序键可以是组合项，但几个排序键之间不能互相重叠。如：

```
01  A.
    02  B.
        03  C PIC X(1).
        03  D PIC 9(5).
```

不能以 B 作为主排序键，又将 C 作为副排序键。

执行本程序后，排好序的记录次序示意如下：

| 日　期 | 顾客号 | 顾客名 | 货　号 | 货　名 | 数　量 | 金　额 |
|---|---|---|---|---|---|---|
| 94　01　08 | 00001029 | ZHANG | 000002 | CAP | 000012 | 00000024.00 |
| 94　12　24 | 00001040 | WANG | 000001 | COAT | 000010 | 00000085.12 |
| 94　04　12 | 00002283 | LI | 000003 | SHIRT | 000100 | 00000700.00 |
| 94　08　5 | 00002283 | LI | 000004 | DESK | 000050 | 00000850.00 |
| 94　08　5 | 00002283 | LI | 000005 | CHAIR | 000070 | 00000728.00 |
| 91　05　8 | 00003756 | TAN | 000003 | SHIRT | 000050 | 00000350.00 |
| 94　04　9 | 00005237 | FUN | 000004 | DESK | 000100 | 00001700.00 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

可以看出，先按顾客号排列，今有二个2283的记录，乃以日期为副排序键，日期相同，再以金额大的排在前面。

SORT 语句的一般格式（之一）

```
SORT 排序中间文件名 ON { ASCENDING  } KEY 数据名1 [，数据名2] …
                       { DESCENDING }
                [ON { ASCENDING  } KEY 数据名3，[，数据名4] …] …
                    { DESCENDING }
    USING 输入文件名
    GIVING 输出文件名
```

“排序中间文件”可以建立在磁带上，也可以建立在磁盘上（它们都是可重用介质），不能建立在非重用的介质上（如打印机文件）。

“输入文件”和“输出文件”必须是顺序文件。要求输入文件的记录区、输出文件的记录区和排序中间文件的记录区三者长度一致。

## §11.5 SORT语句的第二种形式

用上一节介绍的排序语句，只需要一个语句就可以完成全部的排序过程，语句功能强，使用简单。但是它只适用于单纯的排序，即一个输入文件的记录不经任何加工，只是重新编排次序，再形成一个输出文件。

如果我们需要对输入的记录进行某些加工，然后再按指定的某些数据项的顺序排列，那么用上节形式的SORT语句就不行了。我们先举一个实际问题为例来说明它。如果有一批职工的工资记录如下：

| 职工号 | 职工名 | 工资 | 家庭人口 |
|---|---|---|---|
| 0020 | LI | 870 | 3 |
| 0012 | WANG | 1120 | 4 |
| 0087 | ZHANG | 852 | 2 |
| 0032 | TAN | 362 | 1 |
| 0052 | FUN | 888 | 3 |

如果只要求将它们按工资高低排序，则用前面介绍的SORT语句来实现，是比较简单的。因为对这些记录不需作任何加工处理，只是单纯的排序。如果我们现在要求按人口平均收入高低（而不是按个人工资高低）排序，那么，用上述格式的SORT语句是达不到目的的。因为需要先对每个职工算出平均收入（工资/人口），然后再按平均收入排序。也就是说，记录要经过一定的加工处理，并增加一个数据项（平均收入），即每个输入的记录都要计算一次平均收入，然后才能把它放到中间文件中。记录加工和排序后应得到以下顺序：

| 职工号 | 职工名 | 工资 | 人口 | 平均收入 |
|---|---|---|---|---|
| 0087 | ZHANG | 852 | 2 | 426 |
| 0032 | TAN | 362 | 1 | 362 |
| 0052 | FUN | 888 | 3 | 296 |
| 0020 | LI | 870 | 3 | 290 |
| 0012 | WANG | 1120 | 4 | 280 |

对这样的问题，不能使用上一节介绍的办法，而是要先输入记录，进行必要的处理，然后再送到中间文件，进行排序。对记录如何加工处理，根据题目要求而决定。

有的问题更复杂一些，在排好序之后也不是简单地全部复制在输出文件上，而要求在输出之前进行某些处理。例如上面职工工资的例子，如果已按职工家庭人口平均收入排好了序，今要求只将平均收入最高的前一百名的职工记录记入磁盘作输出文件，其它的不存入

磁盘，显然，在输出之前，又需要进行一些处理。

本节介绍另一种格式的 SORT 语句来处理这一类需要在输入、输出过程中进行某些处理的排序任务。

我们先介绍两个在这类排序中用到的辅助的语句。

（一）RELEASE（释放）语句

把中间文件记录区中的记录从内存送到排序中间文件中去。它的作用相当于写（WRITE）语句。但向排序中间文件上写记录，不能用 WRITE 而要用 RELEASE（释放）。意思是从内存释放一个记录到中间文件上去。

它的一般格式是：

RELEASE 记录名 [FROM 标识符]

如果排序中间文件的记录区的名字是 SORT-REC，输入文件的记录区是 IN-REC，则 RELEASE SORT-REC FROM IN-REC。表示从 IN-REC 传送数据给 SORT-REC，再从 SORT-REC 把这个记录送到中间文件上去，见图11.4。

（二）RETURN（回收）语句

从排序中间文件读回（回收）一个记录到内存。见图11.5

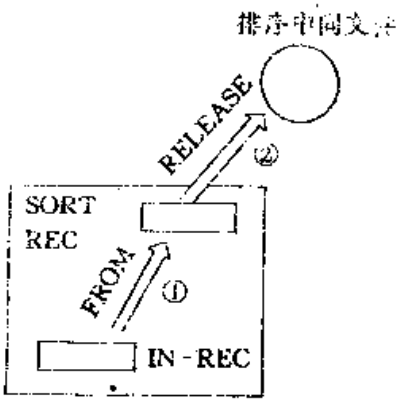



图 11.4

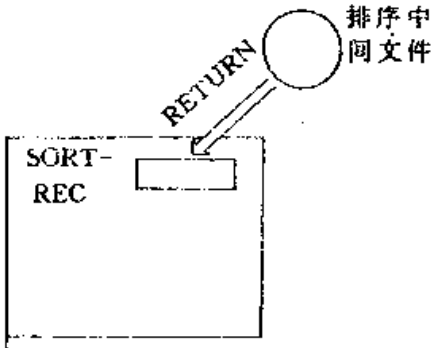



图 11.5

其一般格式为：

RETURN 排序中间文件名 RECORD [INTO 标识符]
[; AT END 强制语句]

它的作用相当于读（READ）语句。但是从排序中间文件读记录不用能 READ 而要用 RETURN。如：

RETURN SORT-WORK FILE AT END GO TO TERM.

当排序中间文件的记录都读完了，就执行 AT END 后面的强制语句。

以上 RELEASE（释放）和 RETURN（回收）都是以内存为主体而言的。从内存⇒中间文件，叫释放。从中间文件⇒内存，叫“回收”。

这两个语句只能在向（从）排序中间文件输出（输入）时使用，并且是在 SORT 语句的控制下进行的。对其它文件不能使用。

【例11.2】 有一批工人月工作时间的记录，记载在磁盘文件上。其格式为：

| WORKER-NUM<br>（工人号） | SHOP-NUM<br>（车间号） | ACTUAL-HOURS<br>（实际工作时间） | OVERTIME<br>（加班时间） |
|---|---|---|---|
| 9（6） | 9（2） | 9（3）V9 | 9（3）V9 |

今要求：

（1）将加班时间≥20小时的工人的记录挑选出来、排选时按车间号由小到大的升序排列。如果是同一车间的，则加班时间多的排在前面。然后存入磁盘文件，需要时可将此磁盘文件打印出来。

（2）将月加班时间≥60小时的职工记录存入磁盘，建立一个“加班积极分子”的磁盘文件。仍以车间号为主排序键，以加班时间为副排序键。

分析题意：要求建立两个不同的磁盘文件。它不是单纯的排序。可以这样进行：输入数据记录后，对加班<20小时的记录不予处理，只将加班≥20的送到排序中间文件去。然后对它进行排序。排好序后把它们逐个记录送到内存，再输出到磁盘，建立第一个磁盘文件。然后再将加班≥60的输出到磁盘，建立第二个磁盘文件。流程图见图11.6。

程序如下：

```
IDENTIFICATION      DIVISION.
PROGRAM-ID.  EXAM11-2.
ENVIRONMENT         DIVISION.
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
        SELECT  INFILE    ASSIGN TO IN-FILE.
        SELECT  SORTFILE    ASSIGN TO SW.
        SELECT  DAFILE   ASSIGN TO DA-FILE.
        SELECT  OUTFILE   ASSIGN TO OUT-FILE.
DATA        DIVISION.
FILE    SECTION.
FD    INFILE  LABEL  RECORD  IS  STANDARD    (输入文件)
      DATA  RECORD  IS  INREC.
01  INREC.
    02  WORKER-NUM       PIC 9(6).
    02  SHOP-NUM         PIC 9(2).
    02  ACTUAL-HOURS     PIC 9(3)V9.
    02  OVERTIME         PIC 9(3)V9.
SD    SORTFILE.                    (排序中间文件)
01  SORTREC.
    02  WORKER-NUM       PIC 9(6).
    02  SHOP-NUM         PIC 9(2).
    02  ACTUAL-HOURS     PIC 9(3)V9.
    02  OVERTIME         PIC 9(3)V9.
FD    DAFILE    LABEL  RECORD  IS  STANDARD.(磁盘文件1)
```

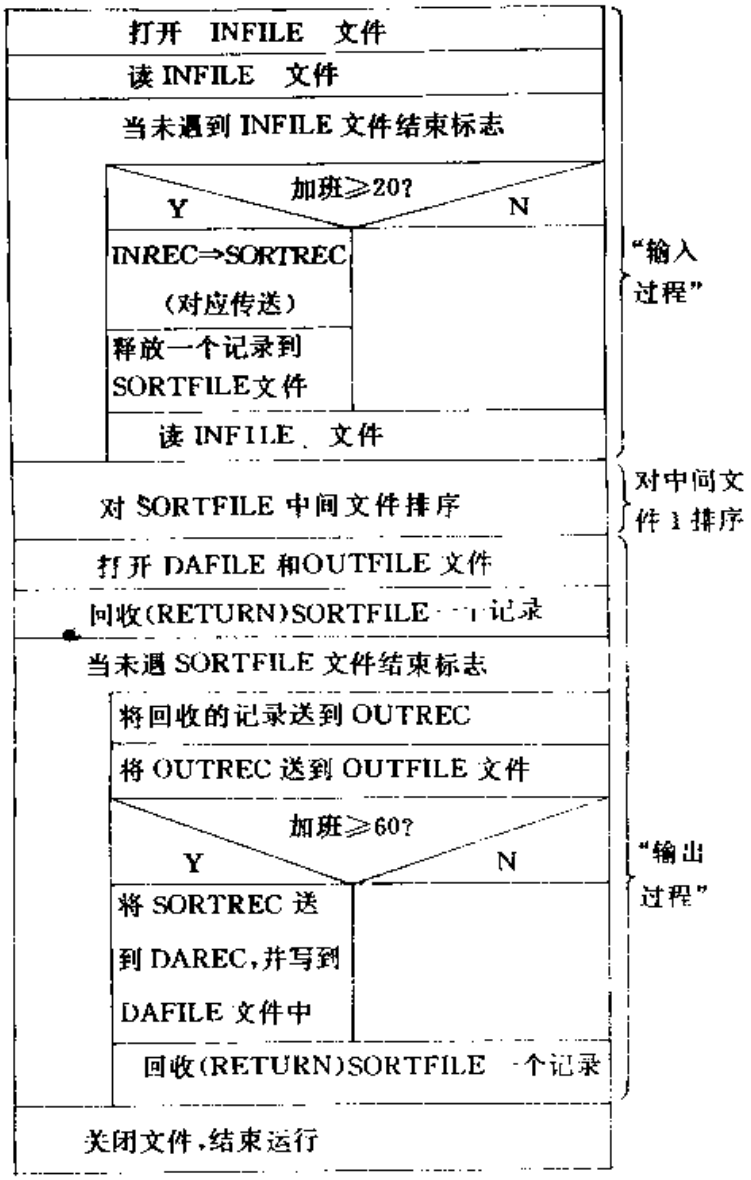



图 11.6

```
01  DAREC.
    02  WORKER-NUM        PIC 9(6).
    02  SHOP-NUM          PIC 9(2).
    02  ACTUAL-HOURS      PIC 9(3)V9.
    02  OVERTIME          PIC 9(3)V9.
FD    OUTFILE    LABEL  RECORD  IS  STANDARD.(磁盘文件2)
01  OUTREC.
    02  FILLER            PIC X(4).
    02  WORKER-NUM        PIC  9(6).
    02  FILLER            PIC X(4).
    02  SHOP-NUM          PIC 9(2).
    02  FILLER            PIC X(4).
    02  ACTUAL-HOURS      PIC Z(3).9.
    02  FILLER            PIC X(4).
    02  OVERTIME          PIC Z(3).9.
PROCEDURE          DIVISION.
```

```
MAIN-PROC  SECTION.
SORT-PROC.
    SORT SORTFILE
        ASCENDING KEY SHOP-NUM OF SORTREC          (主排序键)
        DESCENDING KEY OVERTIME OF SORTREC         (副排序键)
        INPUT PROCEDURE  IS  RECORD SELECTION      (输入过程)
        OUTPUT PROCEDURE IS OUT.                   (输出过程)
    STOP RUN.
RECORD SELECTION  SECTION.                         (输入过程)
OPEN-INFILE.
   OPEN INPUT INFILE.
SELECTION-PROC.
    READ INFILE
          AT END CLOSE INFILE GO TO SELECTION-EXIT.
    IF OVERTIME OF INFILE NOT < 20.0
      MOVE CORR INREC TO SORTREC
      RELEASE    SORTREC.
    GO TO SELECTION-PROC.
SELECTION-END.
    EXIT.
OUT SECTION.                                       (输出过程)
OUT-OPEN.
    OPEN OUTPUT DAFILE,OUTFILE.
RETURN PROC.
    RETURN SORTFILE
        AT END CLOSE DAFILE OUTFILE
        GO TO LIST-AND-OUT-END.
    MOVE SPACE TO OUTREC.
    MOVE CORR SORTREC  TO  OUTREC.
    WRITE  OUTREC  AFTER  2.        (将加班超过20小时的,存放在OUTFILE文件中)
    IF OVERTIME OF SORTREC NOT < 60.0
      WRITE DAREC FROM SORTREC.  (将加班超过60小时的,存放在另一个文件中)
    GO TO RETURN-PROC.
OUT-END.
EXIT.
```

说明：

(1) 过程部的SORT语句与上节介绍的格式不同。不用USING和GIVING短语，而用INPUT PROCEDURE（输入过程）和OUTPUT PROCEDURE（输出过程）。

这个SORT语句是这样执行的：①先执行“输入过程”；②再对中间文件排序；③再执行“输出过程”。INPUT PROCEDURE和OUTPUT PROCEDURE在这里有些类似于PERFORM语句的作用，即转去执行一个“过程”。

本程序中的“输入过程”指定为RECORD SELECTION节（节名是由程序设计者任取

的。为易于理解，今取名的意思是“记录选择节”，即在该节中选择需要的记录送到中间文件去)。开始执行 SORT 语句时，首先转去执行 RECORD-SELECTION 节。在该节中，先打开输入文件，逐个读入记录，将加班20小时以上的记录“释放”到中间文件上去，直到读完全部记录为止。关闭输入文件。

此时，排序中间文件中已是加班20小时以上的职工的记录了，加班20小时以下的记录都已“筛掉”了。

接着，对这个排序中间文件排序。主排序键为车间号（升序），副排序键为加班时间。

最后，应执行“输出过程”。今已指定输出过程是“OUT”节（用这名字也仅是为了易读性。意思是“输出”)。排完序后，转去执行“OUT”节。在这节中，先打开两个输出文件，逐个从排序中间文件中“回收”（可理解为“读入”）已排好序的记录，并输出到磁盘文件 OUT FILE 保存，在需要时可将文件内容打印出来。输出到磁盘前已进行了整编编辑)。且对每个记录中的 OVERTIME（加班时间）进行检查，把加班≥60小时的送到磁盘文件 DAFILE 上去。直到处理完排序中间文件中所有的记录为止。

请注意，在向 DAFILE 写时，不是用 RELEASE（释放）而用 WRITE，读者应弄清楚 WRITE 和 RELEASE，READ 和 RETURN 用法的不同（只有对排序中间文件存取时用 RELEASE 和 RETURN 语句)。

(2)“输入过程”和“输出过程”都必须指定节名，而不是段名。以前一般简单程序的过程部中很少出现“节”，只用到“段”。在这种格式的排序中，过程部必须出现节，例子中用了两个节分别用来作为“输入过程”和“输出过程”。

“输入过程”和“输出过程”的节的位置应放在过程部的 STOP RUN 位置的后面，读者可以看一下本程序是否如此。

“输入过程”和“输出过程”也可以由若干节组成，如 INPUT PROCEDURE IS A THRU B，表示“输入过程”由 A 节到 B 节组成。但输入过程与输出过程不能重叠，如下面的用法是不允许的。

```
A SECTION.  ┐
            │  输入过程(A-B 节)
B SECTION.  ┐
            │  输出过程(B-C 节)
C SECTION.  ┘
```

(3) 在“输入过程”中可以对尚待排序的记录进行加工处理，甚至可以改变排序键的值（但必须在“释放”给中间文件之前)，但这就可能会影响排序的次序了。

排好序后，程序设计者可以通过程序语句对已排好序的记录加工或使用这些记录，但已无法改变这些记录的次序了。

(4)“输入/输出过程”中的语句只能在本过程中控制转移，不能跳出本过程范围之外，也不能从本过程以外转入。如以下用法不正确：

```
A SECTION.          (输入过程)
    GO TO B
B SECTION.          (输出过程)
    GO TO A.
```

SORT 语句的一般格式（之二）为：

```
SORT 排序中间文件名 ON {ASCENDING / DESCENDING} KEY 数据名1 [，数据名2] …
    [ON {ASCENDING / DESCENDING} KEY 数据名3 [，数据名4] …] …
    INPUT PROCEDURE IS 节名1 [{THROUGH / THRU} 节名2]
    OUTPUT PROCEDURE IS 节名3 [{THROUGH / THRU} 节名4]
```

在本节和上一节中介绍了两种形式的SORT 语句。也可以在一个 SORT 语句中混合用“自动输入”（USING）和“输出过程”（OUTPUT PROCEDURE），或混用“输入过程”（*INPUT PROCEDURE*）和“自动输出”。例如，用“输入过程”将输入文件的各记录送到内存，经过加工处理后送到排序中间文件上去，排好序的文件用自动输出（GIVING）方式送到输出文件上去。

应当指出：SORT 语句中必须包括两种输入方式之一和两种输出方式之一。以下五种写法都是不对的：

① SORT SORT-WORK-FILE ASCENDING KEY IS SEX
　　USING IN-FILE.　　（没有“输出方式”）

② SORT SORT-WORK-FILE ASCENDING KEY IS SEX
　　GIVING SORTED-FILE.　　（没有“输入方式”）

③ SORT SORT-WORK-FILE ASCENDING KEY IS SEX
　　INPUT PROCEDURE IS A.　（没有“输出方式”）

④ SORT SORT-WORK-FILE ASCENDING KEY IS SEX
　　OUTPUT PROCEDURE IS B.（没有“输入方式”）

⑤ SORT SORT-WORK FILE ASCENDING KEY IS SEX.　　（没有输入和输出方式）

## §11.6 MERGE（合并）语句

如果有一些已按相同的排序原则排好序的文件，要求将它们合并为一个文件，这叫“合并”或“归并”，用 MERGE 语句来完成。

例如，每一个系的学生已按成绩高低排好名次，分别存在文件1，文件2，…中（每个系的学生记录组成一个文件）。今要求将全校学生按成绩高低排名次，即要求将所有系的学生文件合并成一个大的文件，这个文件也要求按同样的原则排序。

我们可以写：

```
MERGE MERGE-WORK-FILE
  DESCENDING KEY IS AVER
  USING GRADE-FILE-1, GRADE-FILE-2,
```

```
    GRADE-FILE-3, GRADE-FILE-4
  GIVING GRADE-FILE-ALL.
```

它表示将四个系的学生成绩文件（文件名分别定为GRADE-FILE-1到GRADE-FILE-4）合并成一个文件GRADE-FILE-ALL，合并时按每人的平均成绩（AVER）的降序排列。

MERGE语句的一般格式为：

```
MERGE 文件名1 ON
    {ASCENDING / DESCENING} KEY 数据名1 [, 数据名2] …
    [, ON {ASCENDING / DESCENING} KEY 数据名3 [, 数据名4] …]…
    USING 文件名2, 文件名3 [, 文件名4] …
    {OUTPUT PROCEDURE IS 节名1 [{THROUGH / THRH} 节名2]
    {GIVING 文件名5}
```

说明：

(1) MERGE语句中各成分的含义与SORT语句中的基本相同。但应注意，MEGRE不能对未经排序处理的文件进行排序处理。譬如说，如果各系的学生记录未经任何排序（各记录的排列是无规律的），那是不能直接利用MERGE语句来进行“大排队”的。

在执行MERGE语句之前，要求各输入文件中的记录事先已按MERGE语句指出的排序键的原则排列好。譬如在MERGE语句中写“DESCENING KEY IS AVER”，则要求各输入文件事先已按AVER（平均分数）的降序排列好各记录。在执行MERGE语句时，再按此排序键的要求将各文件合并成一个文件，这个输出文件是以AVER的降序排列所有记录的。

(2) MERGE语句不能用输入过程（INPUT PROCEDURE），而只能用自动输入方式（USING）。

(3) 输入文件的个数不得少于两个，最多的个数由各计算机系统规定，读者使用时请注意本系统的规定。

(4)“文件名1”是排序合并的中间工作文件，它应该在数据部中的排序（合并）文件描述体中描述（描述体的层指示符用SD，而不是FD），如果需要保存已合并的结果，应另设一输出文件，“文件5”系统会自动作转存处理。

(5) 待合并的各输入文件（文件名2，文件名3…），输出文件（文件名5）和排序（合并）中间工作文件的记录区大小应该相同。

## 习　题

11.1　什么叫排序？什么叫排序键？

11.2　排序键是否必须是待排序记录中的一项？它是在程序的哪一部分里定义的？

11.3　排序中间工作文件起什么作用?没有它行不行?

11.4　有几种形式的排序语句?它们各有什么特点?

11.5　将本章§11.1给出的“学生情况记录”排序。结果放在磁盘文件中。

（1）按 NAME 升序排序

（2）按年龄 AGE 升序排序。年龄相同则以年级（降序）排序

11.6　按本章§11.5开头给出的职工工资数据，按家庭人口平均收入降序排序，数据由一个磁盘文件提供，结果存在磁盘上。请编写程序。

# 第十二章　报表编制功能

## §12.1　概　　述

众所周知，COBOL 语言最适宜于数据处理，在数据处理中，常常需要编制和输出各种形式的报表，有的报表要求比较高，比较复杂。用我们前面所学过的 COBOL 语言的知识，可以编写出程序，以输出所需的报表。但是，这样做，程序将会复杂、繁琐和冗长，要考虑到许多细节，要在程序中规定每一个步骤如何进行，很容易出错，效率不高。为了满足数据处理中编制报表的特殊需要，COBOL 专门提供了“报表编制”功能，用户只需在程序中提供简单的信息，系统就会按用户的要求输出各种类型的报表，使用十分方便。

COBOL 的“编制报表”的功能很强，规定比较多，有许多细节，如果全部作详细介绍，需要较多篇幅。我们在本章中只是简单介绍“编制报表”功能中最基本、最常用的部分，使读者对其有一概貌性的了解，在深入使用时，可以参阅有关资料。

假设某工厂需要打印出以下格式的生产月报表（虚线左面部分）。

| | |
|---|---|
| 生产月报表 | (报表头 REPORT HEADING) |
| 1994 年 10 月 | (总控制头 CONTROL HEADING FINAL) |
| 单位 产品编号 产品名 数量 单价 金额 | (页头 PAGE HEADING) |
| 1 车间 | (控制头 CONTROL HEADING) |
| A-01 W-12 150 27 4050 | (细目 DETAIL) |
| A-02 P-5 20 300 6000 | |
| ⋮ ⋮ | |
| 小 计 18500 | (控制尾 CONTROL FOOTING) |
| 2 车间 | (控制头 CONTROL HEADING) |
| B-02 F-12 25 500 12500 | (细目 DETAIL) |
| B-10 T-18 100 10 100 | |
| ⋮ ⋮ | |
| 小 计 20832 | (控制尾 CONTROL FOOTING) |
| ⋮ | |
| —01 | (页尾 PAGE FOOTING) |
| ⋮ | |
| （中间还有若干页） | |

| 单位 | 产品编号 | 产品名 | 数量 | 单价 | 金额 |
|---|---|---|---|---|---|
| 12 车间 | | | | | |
| | L-2 | M-2 | 200 | 10 | 2000 |
| | L-4 | U-8 | 120 | 20 | 2400 |
| | ⋮ | | | | ⋮ |
| | 小计 | | | | 8620 |
| | | ---08--- | | | |
| 全部总计 | | | | | 234000 |

工厂生产科制表

(页首 PAGE HEADING)
(控制头 CONTROL HEADING)
(细目 DETAIL)
(控制尾 CONTROL FOOTING)
(页尾 PAGE FOOTING)
(总控制尾 CONTROL FOOTING FINAL)
(报表尾 REPORT FOOTING)

以上是最简单的一种统计报表。要求按车间统计出有关数据，每一页最后有一"页统计"，报表的最后有一项"全部总计"。在每一页的最上面要打印出页头，即各项目名，页的最下部打印出页号。虚线右面部分不是报表的组成部分，只是为了说明 COBOL 编制报表的功能而附加的对照信息，以便在本章的后续部分介绍有关规定时易于对照理解。

应该说明，完全可以不用 COBOL 编制报表的功能来完成以上要求，但程序相当复杂。例如，每一行输出的内容不一样，程序中就要设置不同类型的记录，要考虑换页，以及换页前后的处理等等。我们分析上面的报表，可以看到有以下七种"报表栏"：

1. 报表头；2. 报表尾；3. 页头；4. 页尾；5. 控制头；6. 控制尾；7. 细目。

它们的作用如下：

• "报表头"栏，是整个报表的封面或标题，输出所需要的封面(标题)信息，例如报表名称、报表种类、目的、制表日期等。

• "报表尾"栏，即报表的"封底"，输出报表的结尾信息，如制表单位以及有关说明事项等。

• "页头"栏，输出每一页的标题。这里说的"页"是指"逻辑页"，例如可以指定一个逻辑页为 20 行，每输出 20 行就换页。"逻辑页"和"物理页"(纸的实际大小)可以一致也可以不一致。程序设计者应考虑二者间的关系，使输出整齐美观，易于阅读，易于保存。"逻辑页"的大小是在程序中指定的，如果指定的不合适，会影响报表的使用(例如指定一个逻辑页包括 90 行，超过一个物理页的实际行数，又不足两个物理页，显然是不可取的)。

• "页尾"栏，输出本页的结束信息，如页统计，页号等。

• "控制头"栏，输出统计单位的信息(如车间名、车间号)，由于报表是按单位进行统计的，因此在输出具体的细目行之前应给出该"单位"的信息。可以设置不同层次的单位(如学校、系、班、小组)，分别统计，因此控制头栏可以是嵌套的(分层次的)。

• "控制尾"栏，输出本单位统计数字的小计。

• "细目"栏，报表的具体统计数字，即"明细"。

可以看出，一个单位的信息包括：控制头栏—细目行—控制尾行，如第一车间的标题，第一车间产品细目，第一车间小计。如果一个总公司，下分若干工厂，工厂下设若干车间，则总公司的报表结构为：总公司控制头—工厂控制头—车间控制头—车间产品

细目—车间控制尾—工厂控制尾—总公司控制尾。可以认为，内层(如车间)的控制头—细目—控制尾栏，是其外层单位统计中的“细目”，如：一个车间统计是一个工厂统计的一个“细目”。

图 12.1 表示出整个报表的结构。页首、页尾和控制首、控制尾不是同一概念，它们在逻辑上是无关的。

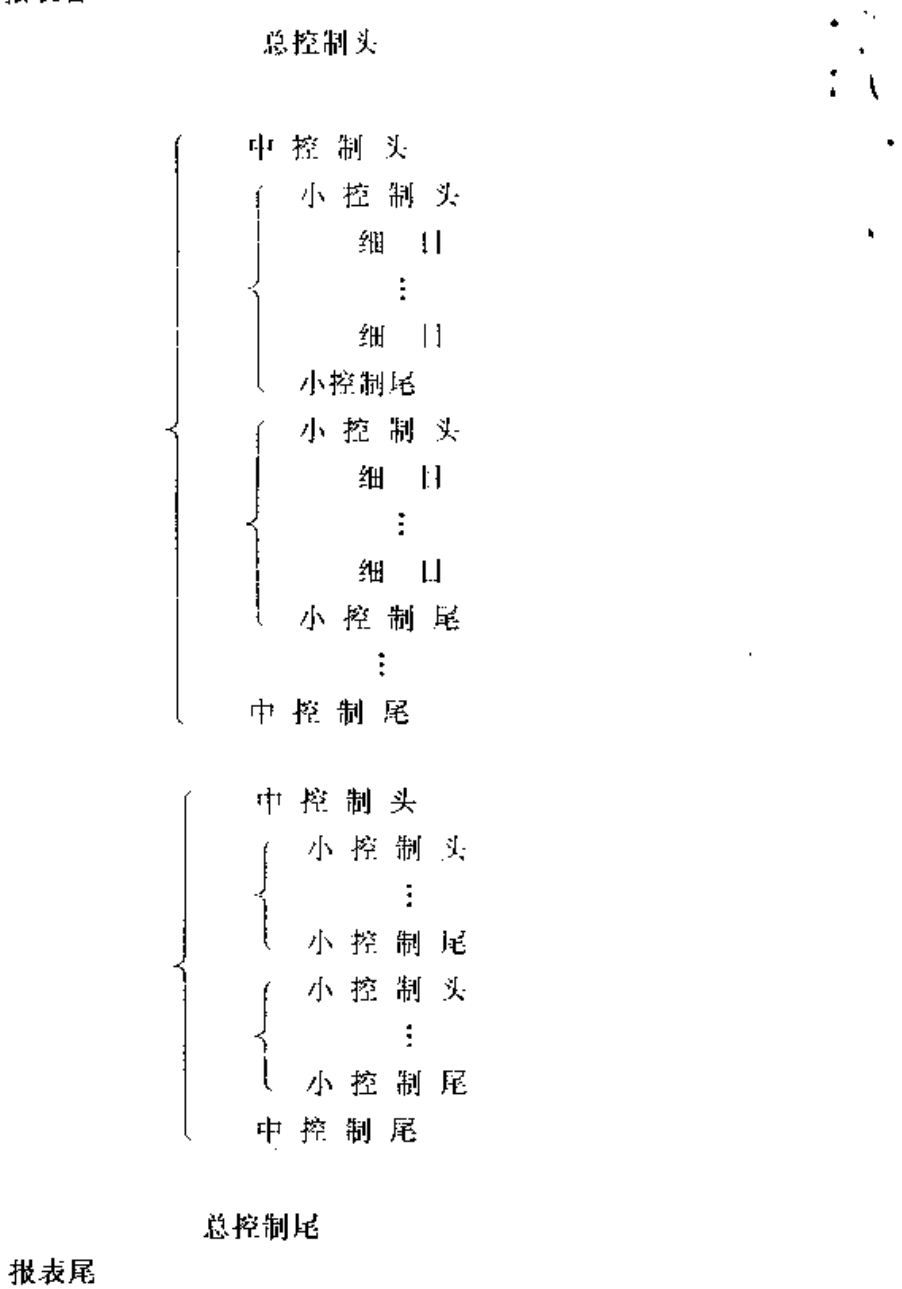



图 12.1

## § 12.2 报表编制功能在 COBOL 程序中的描述

### 12.2.1 在数据部中的描述

报表编制功能的重点是对于整个报表的逻辑描述，因此在数据部中应作详尽的描述。过程部的有关成分却相当简单。

(一) 文件节

在 FILE SECTION 中，要对报表所在的文件进行描述。其一般格式为：

```
FD  文件名
    LABEL  {RECORDS  ARE}STANDARD
           {RECORD   IS }
    [BLOCK CONTAINS [整数 1 TO] 整数 2{RECORDS   }]
                                      {CHARACTERS}
    [RECORD CONTAINS [整数 3 TO] 整数 4  CHARACTERS]
    [DATA{RECORD  IS }数据名 1 [，数据名 2] …]
         {RECORDS ARE}
    [VALUE OF 数据名 3 IS {数据名 4}[，数据名 4 IS{数据名 5}]…]
                          {常量 1  }             {常量 2  }
    [CODE CHARACTER]
    {REPORT   IS }报表名 1 [，报表名 2] …
    {REPORTS  ARE}
```

可以看到，与以前介绍过的相比，多了一个 REPORT 子句。

作为报表文件，描述体中必须有 REPORT 子句，但不能有记录描述体。

(二) 报表节

对报表编制功能的详细描述，是由数据部中的报表节来完成的。

报表节的一般格式如下：

```
REPORT   SECTION
    [RD  报表名
    [CODE  字符串]
    [{CONTROL   IS }{数据名 1 [，数据名 2] …                    }]
     {CONTROLS  ARE}{FINAL [，数据名 1 [，数据名 2] …]}
    [PAGE[LIMIT IS ]整数 1[LINE ] [，HEADING 整数 2]
         [LIMITS ARE]     [LINES]
    [FIRST  DETAIL 整数 3] [，LAST  DETAIL 整数 4]
    [，FOOTING  整数 5]]
    {报表栏描述体} …] …
```

这里的“报表名”就是在文件节中相应的 FD 文件描述体内的 REPORT 子句所指定的报表名。一个文件可以有若干个报表名，一个报表名可以有若干个报表描述体。报表名在程序中必须是唯一的。

CODE 子句的作用是区别不同的报表。因为一个文件可以有若干个报表名。在 CODE 后面写 1～2 个字符(即指定一个字符串)，通过此字符串来标识不同的报表。例如，“A1”表示一个报表，“A2” 代表另一个报表。

PAGE 子句的作用是定义逻辑页的长度以及各报表栏在页中的纵向定位。

整数 1——一页中允许打印的最大行数。

整数 2——报表首栏或页首栏的第一行的位置，例如可以规定报表首栏从第 3 行开始打印。

整数 3——表示能出现报表体(控制首栏，控制尾栏或细目栏)的位置。例如可以规定控制首栏从第 5 行开始输出。

整数 4——表示能出现控制首栏或细目栏的最后一行的位置。例如规定以上各栏打印不得超过本页的第 35 行。

整数 5——指定控制尾栏的最后一行不得超过的位置。如未指定控制尾栏的内容不应超过第 50 行。

页尾和报表尾必须在整数 5 规定的位置之下。一个逻辑页的结构如图 12.2 所示。

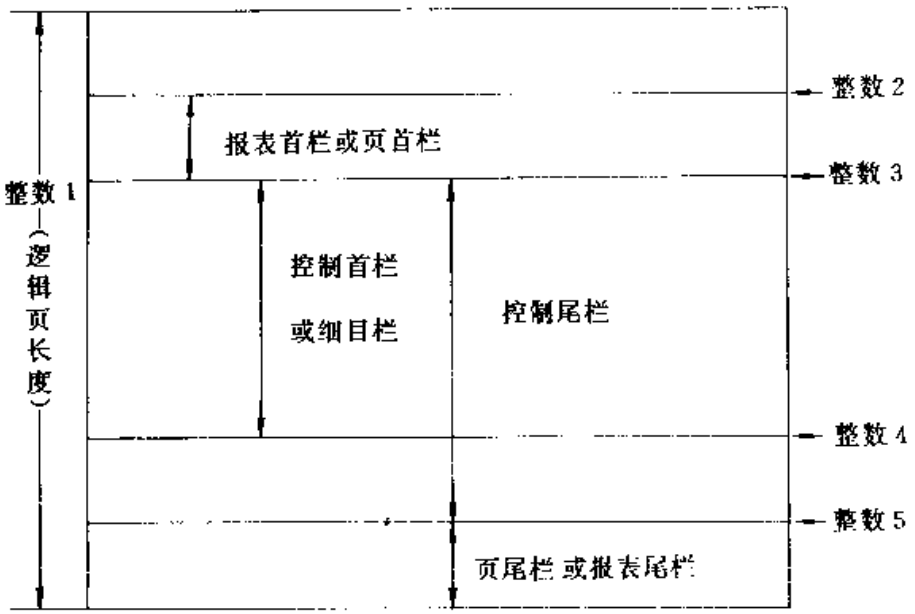



图 12.2

若 PAGE 子句缺省，则系统按规定的页长标准处理。

CONTROL 子句的作用是确定报表的控制层次。如果 CONTROL 子句为“CONTROL FINAL，FACTORY，WORKSHOP”，表示：FINAL 是最外层(即最高层)控制，FACTORY(工厂)是第二层，WORKSHOP(车间)是第三层。也就是按总控制—FACTORY(工厂)—WORKSHOP(车间)的次序嵌套，按层次进行控制。CONTROL 子句指定：在连续打印输出细目行时遇到什么情况应暂停输出细目行以便进行所需要的“阶段处理”，这叫“控制截断”。例如当处理完第一个车间的细目行以后，不是连续再输出第二个车间的细目行，而是转去进行车间小计输出第二车间的“控制头”栏等“阶段处理”，然后再接着输出第二车间的细目行。当一个工厂全部车间的细目行输出完以后，应进行车间小计、工厂小计、打印下一个工厂的控制头栏、车间控制头栏，然后再接着输出下一工厂第一车间的细目行。有关控制截断是如何进行的，可参考本章例 12.1 中有关说明。

下面介绍报表栏描述体。在报表节 (REPORT SECTION) 中可以看到：一个报表可以包括多个报表栏描述体。前面已介绍，一共有七种报表栏(报表头、报表尾、页头、页尾、控制头、控制尾、细目)。每一个报表栏的内容都要用一个“报表栏描述体”来描述(即定

义)。报表栏描述体的一般格式如下：

```
01 [数据名 1]
    [LINE NUMBER IS {整数 1 [ON NEXT PAGE]
                    {PLUS 整数 2}]
    [; NEXT GROUP IS {整数 3
                     {PLUS 整数 4
                     {NEXT PAGE}]
                    {REPORT HEADING}
                    {RH}
                    {PAGE HEADING}
                    {PH}
                    {CONTROL HEADING} {数据名 2}
                    {CH}              {FINAL}
    TYPE IS         {DETAIL}
                    {DE}
                    {CONTROL FOOTING} {数据名 3}
                    {CF}              {FINAL}
                    {PAGE FOOTING}
                    {PF}
                    {REPORT FOOTING}
                    {RF}
                    [; [USAGE IS] DISPLAY]
```

层号 01 表示这是报表栏描述体。它相当一个记录。一个报表栏由若干个报表栏元素组成(如同一个记录由若干个数据项组成一样)。在报表栏描述体后面应跟有报表栏元素的描述体。

用 LINE NUMBER 子句指定报表栏在纵向的定位信息。整数 1 表示一个逻辑页内的绝对行号，整数 2 表示相对行号。NEXT 短语表示在下页出现相应的报表栏。用 NEXT GROUP 子句指定该报表栏后面一个报表栏在什么位置。TYPE 子句用来表明该报表栏的类型(如细目栏或控制首栏等)。此子句必须存在。

USAGE 子句用来说明该报表栏中的元素为打印项。

如同一个记录包含若干组合项，一个组合项又包含若干基本项一样，一个报表栏由若干“报表栏元素组”构成，而一个报表栏元素组又包含若干个“元素项”。后者的层号应比前者大。

“报表栏元素组”描述体的一般格式如下：

层号 [数据名 1]

[LINE NUMBER IS {整数 1 [ON NEXT PAGE] / PLUS 整数 2}]

[; [USAGE IS] DISPLAY]。

"报表栏元素项"描述体的一般格式如下：

层号 [数据名 1]

[BLANK WHEN ZERO] [GROUP INDICATE]

{JUSTIFIED / JUST} RIGHT

[LINE NUMBER IS {整数 1 [ON NEXT PAGE] / PLUS 整数 2}]

[COLUMN NUMBER IS 整数 3] {PICTURE / PIC} IS 字符串

{SOUREC IS 标识符 1 / VALUE IS 常量 / SUM 标识符 2 [，标识符 3] … [UPON 数据名 [，数据名 3] …]) … / RESET ON {数据名 4 / FINAL}]}

[ [USAGE IS] DISPLAY]。

报表栏元素组描述体的层号范围为 02～48。USAGE 子句说明至少有一个下属报表栏元素项为打印项。

报表栏元素项描述体的层号范围为 02～49。一个报表栏描述体可以由多层组成，用层号来表示它们之间的隶属关系。一个报表的层次关系可以表示为：

报表—报表栏—报表栏元素组—报表栏元素项。

报表栏元素项描述体中的 USAGE 子句说明该报表栏元素项为打印项。LINE NUMBER 用来纵向定位。BLANK WHEN ZERO 子句、PICTURE 子句和 JUSTIFIED 子句的含义与以前介绍过的相同。

COLUMN 子句对报表栏元素项进行横向定位。即指定列的位置。COLUMN 子句只能用于报表栏元素项，同时必须有一个 LINE NUMBER 子句与之匹配使用。在同一行上，各打印项应按照列号递增的顺序来定义。整数 3 表示该打印项的起始列号。

打印项的值是由 SOURCE 子句，VALUE 子句和 SUM 子句指定的。SOURCE 子句指定发送数据项。规定将数据项的值传送到相应的打印项中，相当于执行一个 MOVE 语句。"标识符 1" 可以是数据部中任何节内定义的一个数据项，也可以是页计数器、行计数器或求和计数器。VALUE 子句表明发送项是一个常量(即 VALUE 后给出的常量)。SUM 子句的含义是根据报表功能的要求对数据项进行求和。其使用比较复杂，在此不作专门介绍，可以结合例 12.1 程序了解其使用方法。

### 12.2.2 在过程部中的描述

*(一) INITIATE 语句(初始化语句)*

格式如下：

```
INITIATE    报表名1  [，报表名2] …
```

其中报表名1，报表名2…必须已在报表节中定义过。此语句的作用是使这些报表的求和计数器和行计数器置零，页计数器置1。

系统为每一个报表设两个计数器：PAGE-COUNTER（页计数器）和LINE-COUNTER（行计数器）。初始化对它们置以初值，以后用来累计页数和行数。其值可以被过程部语句引用。

（二）GENERATE语句（生成语句）

格式如下：

```
GENERATE { 数据名 }
         { 报表名 }
```

GENERATE语句按照数据部报表节规定的报表描述产生相应的报表栏和其它操作。“数据名”必须是细目报表栏名，并且可以用报表名来对其限定。如果使用“报表名”，则不打印细目报表栏，仅进行求和等操作。

（三）TERMINATE语句（终止语句）

格式如下：

```
TERMINATE  报表名1 [，报表名2] …
```

此语句使报表编制控制系统对指定的报表完成所有的结束处理，产生报表尾栏。它不关闭文件。关闭文件的操作由CLOSE语句完成。

## §12.3 报表编制功能应用举例

**【例12.1】** 利用COBOL“报表打印”功能来统计和打印学生成绩数据。

输入的数据为每个学生的学号（为6位数字，前二位代表系，中间二位代表班，最后二位代表学生在班中序号。如010204表示01系02班04号学生）、姓名、性别、年龄、五门课的成绩。

要求：

（1）分班打印出每个学生以上数据，并统计前三门课（主课）总分、后二门课（辅课）总分、五门课总分、平均分数，并将需要补考的（有一门以上不合格）注出来。

（2）按班统计：全班人数、班中女生人数、平均年龄、各门课分别的平均成绩、全班总平均成绩、全班需补考人数。

（3）按系统计以上数据（项目与班统计相同）。

（4）按校统计以上数据（项目与系班统计相同）。

（5）要求设计打印报表头、页头、小标题等，使打印格式清楚、美观。

程序如下：

```
001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID.  EXAM 12-1
001030 ENVIRONMENT  DIVISION.
001040 INPUT-OUTPUT  SECTION.
001050 FILE-CONTROL.
001060 SELECT  DATA-FILE  ASSIGN  TO  FILE1.(输入文件)
001070 SELECT  REPORT-FILE  ASSIGN  TO  打印机.
001080 DATA  DIVISION.
001090 FILE  SECTION.(文件节)
001100 FD  DATA-FILE  LABEL  RECORD  IS.  STANDARD.
001110 01  DATA-REC.
001120   02  NUMB.(学号)
001130     03  DEPART(系)  PIC    99.
001140     03  CLASS(班)  PIC  99.
001150     03  NUM(序号)  PIC  99.
001160   02  NAME-OF-STUDENT(姓名)  PIC  X(20).
001170   02  SEX(性别)  PIC  X.
001180   02  AGE(年龄)  PIC  99.
001190   02  COURS(课程)  OCCURS  5  PIC  999V9.
001200
002010 FD  REPORT-FILE
002020     LABEL  RECORD  IS  OMITTED
002030                REPORT  IS  REP.(报表名为 REP)
002040     WORKING-STORAGE  SECTION.      (工作单元节)
002050     77     N              PIC          999  VALUE  1.
002060     77     SUM1           PIC          999V9.
002070     77     SUM2           PIC          999V9.
002080     77     TOTAL          PIC          999V9.
002090     77     AVER           PIC          999V9.
002100     77     AVER-C         PIC          999V9.
002110     77     AVER-D         PIC          999V9.
002115     77     AGE-AVE-U      PIC          99V9.
002120     77     SEX-W          PIC          9.
002130     77     PASS           PIC          XX.
002140     77     PASS-W         PIC          9.
002150     77     AGE-AVE-C      PIC          99V9.
002160     77     AGE-AVE-D      PIC          99V9.
002168     77     AVER-U         PIC          9(3)V9.
002170     01     AV.
002180       02  AVE       OCCURS     5  PIC  999V9.
002190  REPORT     SECTION.                (报表节)
002200  RD     REP
003010       CONTROLS     ARE       FINAL     DEPART     CLASS
```

```
003020      PAGE          LIMITS   ARE      66          LINES
003030      HEADING       3
003040      FIRST         DETAIL   9
003050      LAST          DETAIL   52
003060      FOOTING       61.
003070 01  REP-HEAD  TYPE    IS  REPORT  HEADING
003080          NEXT  GROUP  IS  NEXT  PAGE.
003090  02  LINE    15  COLUMN  40  PIC  X(50)
003100      VALUE  IS  'THE  REPORT  OF  THE  STUDENT  EXAMINATIONS'.
003110  02  LINE    50  COLUMN  45      PIC      X(40)
003120        VALUE  IS  'QINGHUA  UNIVERSITY  1993.1'.
003130 01  PAGE HEAD  TYPE  IS  PAGE  HEADING.
003140    02    LINE    4
003150      • COLUMN  50  PIC          X(30)
003160        VALUE    IS  'QINGHUA UNIVERSITY'.
003170    02  LINE       6
003180        COLUMN  45  PIC          X(50)
003190        VALUE    IS  'THE REPORT OF THE STUDENT EXAMINATIONS'.
003200    02  LINE       7.
004010      03    COLUMN    110  PIC  X(4)  VALUE    'PAGE'.
004020      03    COLUMN    115  PIC  ZZ9  SOURCE    PAGE-COUNTER.
004030  01  DEPART-HEAD    TYPE  IS  CONTROL  HEADING  DEPART
004040          NEXT      GROUP    PLUS  2.
004050      02    LINE          PLUS  2.
004060      03    COLUMN  55  PIC  X(12)  VALUE    'DEPARTMENT'.
004070      03    COLUMN  70  PIC  99     SOURCE    DEPART.
004080 *
004090  01  CLASS-HEAD  TYPE  IS  CONTROL  HEADING  CLASS
004100          NEXT  GROUP  PLUS  1.
004110    02      LINE    PLUS  1
004120            COLUMN  1  PIC  X(136)  VALUE  ALL  '-'.
004130    02      LINE    PLUS  1.
004140      03  COLUMN    2    PIC  X(6)    VALUE    'DEPART'.
004150      03  COLUMN    12   PIC  X(5)    VALUE    'CLASS'.
004160      03  COLUMN    20   PIC  X(6)    VALUE    'NUMBER'.
004170      03  COLUMN    28   PIC  X(3)    VALUE    'SEX'.
004180      03  COLUMN    33   PIC  X(3)    VALUE    'AGE'.
004190      03  COLUMN    38   PIC  X(6)    VALUE    'COUR-1'.
004200      03  COLUMN    48   PIC  X(6)    VALUE    'COUR-2'.
005010      03  COLUMN    58   PIC  X(6)    VALUE    'COUR-3'.
005020      03  COLUMN    68   PIC  X(6)    VALUE    'COUR-4'.
005030      03  COLUMN    78   PIC  X(6)    VALUE    'COUR-5'.
005040      03  COLUMN    90   PIC  X(6)    VALUE    'SUM-1'.
```

```
005050       03  COLUMN    100  PIC  X(6)     VALUE    'SUM-2'.
005060       03  COLUMN    110  PIC  X(5)     VALUE    'TOTAL'.
005070       03  COLUMN    120  PIC  X(4)     VALUE    'AVER'.
005080       03  COLUMN    130  PIC  X(4)     VALUE    'PASS'.
005090 *
005100  01  DETAIL-LINE    TYPE    IS    DETAIL.
005110      02    LINE    PLUS    1.
005120      03    COLUMN    4    PIC  99        GROUP  SOURCE  DERART.
005130      03    COLUMN    14   PIC  99        GROUP  SOURCE  CLASS.
005140      03    COLUMN    20   PIC  9 (6)           SOURCE  NUMB!
005150      03    COLUMN    29   PIC  X                SOURCE  SEX.
005160      03    COLUMN    34   PIC  99        SOURCE  AGE.
005170      03    COLUMN    39   PIC  ZZZ. 9    SOURCE  COURS  (1).
005180      03    COLUMN    49   PIC  ZZZ. 9    SOURCE  COURS  (2).
005190      03    COLUMN    59   PIC  ZZZ. 9    SOURCE  COURS  (3).
005200      03    COLUMN    69   PIC  ZZZ. 9    SOURCE  COURS  (4).
006010      03    COLUMN    79   PIC  ZZZ. 9    SOURCE  COURS  (5).
006020      03    COLUMN    90   PIC  ZZZ. 9    SOURCE  SUM1.
006030      03    COLUMN    100  PIC  ZZZ. 9    SOURCE  SUM2.
006040      03    COLUMN    110  PIC  ZZZ. 9    SOURCE  TOTAL.
006050      03    COLUMN    120  PIC  ZZZ. 9    SOURCE  AVER.
006060      03    COLUMN    131  PIC  XX        SOURCE  PASS.
006070 *
006080  01  CLASS COUNT    TYPE  IS  CONTROL    FOOTING  CLASS
006090              NEXT    GROUP    PLUS    3.
006100      02  LINE    PLUS  2
006110          COLUMN  1    PIC  X (136) VALUE  ALL  ' '.
006120      02  LINE    PLUS  1.
006130      03  NUM-OF-CLASS COLUMN  22  PIC  ZZ9  SUM  N.
006140      03  NUM OF-FEM-C COLUMN  27  PIC  ZZ9    SUM  SEX W.
006150      03  AGE-TOTAL-C                PIC    999     SUM  AGE.
006160      03  COLUMN  34                 PIC    99.9    SOURCE  AGE-AVE-C.
006170      03  TOTAL-1-C                  PIC  9999V9    SUM  COURS  (1).
006180      03  TOTAL-2-C                  PIC  9999V9    SUM  COURS  (2).
006190      03  TOTAL-3-C                  PIC  9999V9    SUM  COURS  (3).
006200      03  TOTAL-4 C                  PIC  9999V9    SUM  COURS  (4).
007010      03  TOTAL-5 C                  PIC  9999V9    SUM  COURS  (5).
007020      03  TOTAL-ALL-C                PIC  9(5)V9    SUM  TOTAL.
007030      03  COLUMN  39                 PIC  ZZZ.99    SOURCE  AVE  (1).
007040      03  COLUMN  49                 PIC  ZZZ.99    SOURCE  AVE  (2).
007050      03  COLUMN  59                 PIC  ZZZ.99    SOURCE  AVE  (3).
```

```
007060    03 COLUMN  69                    PIC   ZZZ.99   SOURCE  AVE  (4).
007070    03 COLUMN  79                    PIC   ZZZ.99   SOURCE  AVE  (5).
007080    03 COLUMN  120                   PIC   ZZZ.99   SOURCE  AVER-C.
007090    03 PASS-C  COLUMN     131        PIC     Z9     SUM  PASS-W.
007100 01    DEPART-COUNT  TYPE  IS  CONTROL  FOOTING  DEPART
007110         NEXT  GROUP  NEXT  PAGE.
007120    02   LINE    53  COLUMN  2  PIC  X(134)  VALUE  ALL  '-'.
007130    02   LINE    PLUS  2
007140         COLUMN  10  PIC  X(25)  VALUE  'THE DEPARTMENT COUNT:'.
007150    02   LINE    PLUS  2.
007160      03  COLUMN  2    PIC  X(6)  VALUE  'DEPART'.
007170      03  COLUMN  20   PIC  X(6)  VALUE  'NUMBER'.
007180      03  COLUMN  28   PIC  X(3)  VALUE  'SEX'.
007190      03  COLUMN  33   PIC  X(3)  VALUE  'AGE'.
007200      03  COLUMN  38   PIC  X(6)  VALUE  'COUR-1'.
008010      03  COLUMN  48   PIC  X(6)  VALUE  'COUR-2'.
008020      03  COLUMN  58   PIC  X(6)  VALUE  'COUR-3'.
008030      03  COLUMN  68   PIC  X(6)  VALUE  'COUR-4'.
008010      03  COLUMN  78   PIC  X(6)  VALUE  'COUR-5'.
008050      03  COLUMN  120  PIC  X(4)  VALUE  'AVER'.
008060      03  COLUMN  130  PIC  X(4)  VALUE  'PASS'.
008070    02    LINE    PLUS  2.
008080      03  COLUMN  4            PIC  99      SOURCE DEPART.
008090      03  NUM OF DEPART  COLUMN  22  PIC    ZZ9    SUM NUM OF CLASS.
008100      03  NUM-OF-FEM-D   COLUMN  27  PIC    ZZ9    SUM NUM-OF-FEM-C.
008110      03  AGE-TOTAL D                  PIC  9999     SUM AGE-TOTAL C.
008120      03  COLUMN  34                   PIC  99.9     SOURCE AGE AVE D.
008130      03  TOTAL-1-D                    PIC  9 (4) .9 SUM  TOTAL-1-C.
008140      03  TOTAL-2 D                    PIC  9 (4) .9 SUM  TOTAL 2-C.
008150      03  TOTAL-3-D                    PIC  9 (4) .9 SUM  TOTAL-3 C.
008160      03  TOTAL-4-D                    PIC  9 (4) .9 SUM  TOTAL 4 C.
008170      03  TOTAL 5 D                    PIC  9 (4) .9 SUM  TOTAL-5-C.
008180      03  COLUMN  39      PIC  ZZZ.99  SOURCE  AVE  (1).
008190      03  COLUMN  19      PIC  ZZZ.99  SOURCE  AVE  (2).
008200      03  COLUMN  59      PIC  ZZZ.99  SOURCE  AVE  (3).
009010      03  COLUMN  69      PIC  ZZZ.99  SOURCE  AVE  (4).
009020      03  COLUMN  79      PIC  ZZZ.99  SOURCE  AVE  (5).
009030      03  TOTAL-ALL-D     PIC  9(4).9  SUM  TOTAL-ALL-C.
009040      03  COLUMN  120  PIC  9(2).99  SOURCE  AVER-D.
009050      03  PASS-D
009060        COLUMN  131  PIC  ZZ9  SUM  PASS-C.
009070    02    LINE  PLUS  2  COLUMN  50  PIC  X(30)  VALUE  ALL  '*'.
```

```
009080 01  UNIV-COUNT  TYPE  IS  CONTROL  FOOTING  FINAL.
009090  02     LINE  20
009100         COLUMN  50  PIC  X(30)  VALUE 'ALL  DEPARTMENT  WAS  PRO-
CESSED'.
009110  02     LINE  PLUS  3
009120         COLUMN  2  PIC  X(134)  VALUE  ALL  '-'.
009130  02     LINE  PLUS  2.
009140    03   COLUMN  2  PIC  X(24)  VALUE  'NUMBER  OF  ALL  UNIVERSITY'.
009150    03   COLUMN  28  PIC  X(3)  VALUE  'SEX'.
009160    03   COLUMN  33  PIC  X(3)  VALUE  'AGE'.
009170    03   COLUMN  38  PIC  X(6)  VALUE  'COUR-1'.
009180    03   COLUMN  48  PIC  X(6)  VALUE  'COUR-2'.
009190    03   COLUMN  58  PIC  X(6)  VALUE  'COUR 3'.
009200    03   COLUMN  68  PIC  X(6)  VALUE  'COUR 4'.
010010    03   COLUMN  78  PIC  X(6)  VALUE  'COUR-5'.
010020    03   COLUMN  120  PIC  X(4)  VALUE  'AVER'.
010030    03   COLUMN  130  PIC  X(1)  VALUE  'PASS'.
010040  02     LINE PLUS  2.
010050    03   NUM-OF-UNIV  COLUMN    12     PIC  ZZZ9  SUM  NUM-OF-DEPART.
010060    03   NUM-OF-FEM-U  COLUMN  27     PIC  ZZ9    SUM  NUM OF FEM-D.
010070    03   AGE TOTAL-U                  PIC  9(5)  SUM       AGE-TOTAL-D.
010080    03   COLUMN  34  PIC  99.9  SOURCE  AGE-AVE-U.
010090    03   TOTAL 1 U  PIC  9(4).9  SUM  TOTAL-1-D.
010100    03   TOTAL-2-U  PIC  9(4).9  SUM  TOTAL-2-D.
010110    03   TOTAL-3-U  PIC  9(4).9  SUM  TOTAL-3-D.
010120    03   TOTAL 4 U  PIC  9(4).9  SUM  TOTAL-4 D.
010130    03   TOTAL-5-U  PIC  9(4).9  SUM  TOTAL-5-D.
010140    03   COLUMN  39  PIC  ZZZ.9  SOURCE  AVE  (1).
010150    03   COLUMN  49  PIC  ZZZ.9  SOURCE  AVE  (2).
010160    03   COLUMN  59  PIC  ZZZ.9  SOURCE  AVE  (3).
010170    03   COLUMN  69  PIC  ZZZ.9  SOURCE  AVE  (4).
010180    03   COLUMN  79  PIC  ZZZ.9  SOURCE  AVE  (5).
010190    03   TOTAL-ALL-U  PIC  9(4).9  SUM  TOTAL-ALL-D.
010200    03   COLUMN  120  PIC  9(2).9  SOURCE  AVER-U.
011010    03   PASS-U  COLUMN  131  PIC  ZZ9  SUM  PASS-D.
011020  02     LINE  PLUS  2
011030         COLUMN  2  PIC  X(134)  VALUE  ALL  ' '.
011040 *
011050 *
011060 01  REPORT-END    TYPE  IS  REPORT  FOOTING.
011070  02     LINE  25  NEXT  PAGE.
011080    03   COLUMN  50  PIC  X(14)  VALUE  'REPORT    END'.
011090  02     LINE  35.
```

```
011100     03  COLUMN  50  PIC  X(10)  VALUE  'WRITTER  :'.
011110     03  COLUMN  60  PIC  X(10)  VALUE  'LI-FUN'.
011120 *
011130 *
011140 *
011150  PROCEDURE       DIVISION.
011160  DECLARATIVES.
011170  D-1       SECTION.
011180         USE  BEFORE  REPORTING  CLASS-COUNT.
011190  P1.    DIVIDE    AGE-TOTAL-C  BY  NUM-OF-CLASS  GIVING  AGE-AVE-C.
011200         DIVIDE    TOTAL-1-C    BY  NUM-OF-CLASS  GIVING  AVE  (1).
012010         DIVIDE    TOTAL-2-C    BY  NUM-OF-CLASS  GIVING  AVE  (2).
012020         DIVIDE    TOTAL-3-C    BY  NUM-OF-CLASS  GIVING  AVE  (3).
012030         DIVIDE    TOTAL-4-C    BY  NUM-OF-CLASS  GIVING  AVE  (4).
012040         DIVIDE    TOTAL-5-C    BY  NUM-OF-CLASS  GIVING  AVE  (5).
012050         COMPUTE AVER-C=TOTAL-ALL-C/ (5*NUM-OF-CLASS).
012060 *
012070  D-2       SECTION.
012080       USE  BEFORE  REPORTING  DEPART-COUNT.
012090  P2.  DIVIDE    AGE-TOTAT-D  BY  NUM-OF-DEPART  GIVING  AGE-AVE D.
012100       DIVIDE    TOTAL-1-D    BY  NUM-OF-DEPART  GIVING  AVE  (1).
012110       DIVIDE    TOTAL-2-D    BY  NUM-OF-DEPART  GIVING  AVE  (2).
012120       DIVIDE    TOTAL-3-D    BY  NUM-OF-DEPART  GIVING  AVE  (3).
012130       DIVIDE    TOTAL-4-D    BY  NUM-OF-DEPART  GIVING  AVE  (4).
012140       DIVIDE    TOTAL-5-D    BY  NUM-OF-DEPART  GIVING  AVE  (5).
012150       COMPUTE  AVER-D=TOTAL ALL-D/ (5*NUM-OF-DEPART).
012160 *
012170  D 3     SECTION.
012180    USE  BEFORE  REPORTING  UNIV-COUNT.
012190    DIVIDE  AGE-TOTAL-U  BY  NUM-OF-UNIV  GIVING  AGE-AVE-U.
012200    DIVIDE  TOTAL-1-U    BY  NUM-OF-UNIV  GIVING  AVE  (1).
013010    DIVIDE  TOTAL-2-U    BY  NUM OF-UNIV  GIVING  AVE  (2).
013020    DIVIDE  TOTAL 3-U    BY  NUM-OF-UNIV  GIVING  AVE  (3).
013030    DIVIDE  TOTAL 4-U    BY  NUM-OF-UNIV  GIVING  AVE  (4).
013040    DIVIDE  TOTAL 5-U    BY  NUM-OF-UNIV  GIVING  AVE  (5).
013050    COMPUTE  AVER-U  TOTAL-ALL-U/ (5*NUM-OF-UNIV).
013060  END     DECLARATIVES.
013070 *
013080  S       SECTION.
013090  STA.
013100       OPEN  INPUT  DATA-FILE, OUTPUT  REPORT-FILE.
013120       INITIATE  REP.
013130  PROC.
```

```
013140      READ DATA-FILE AT END GO TO FINISH.
013150      ADD COURS (1) COURS (2) COURS (3) GIVING SUM1.
013160      ADD COURS (4) COURS (5)           GIVING SUM2.
013170      ADD SUM1 SUM2 GIVING TOTAL.
013180      DIVIDE TOTAL BY 5 GIVING AVER ROUNDED.
013190      IF (COURS (1) < 60) OR (COURS (2) <60)
013200      OR(COURS(3)<60) OR (COURS (4)<60) OR (COURS (5) <60)
014010            MOVE 'NO' TO PASS MOVE 1 TO PASS-W
014020      ELSE MOVE ' ' TO PASS MOVE 0 TO PASS-W.
014030      IF SEX='F' MOVE 1 TO SEX-W
014040      ELSE       MOVE 0 TO SEX-W.
014050    GENERATE DETAIL-LINE.
014060    GO TO PROC.
014070  FINISH.
014080      TERMINATE REP.
014090      CLOSE DATA-FILE
014100        REPORT-FILE.
014110      STOP RUN.
```

说明：

(1) 在环境部中定义一个输入文件DATA-FILE，从它输入学生的数据。在数据部的文件节中(001100行开始)对它的数据结构进行描述。在DATA-REC记录中(001190行)，设立一个“表”COURS(COURSE的缩写，代表“课程”)，内含五个元素分别存放五门课成绩，这和过去学过的没什么不同。

(2) 在环境部中还定义一个REPORT-FILE文件作为报表输出文件，指定在宽行打印机输出。这个文件名是用户任意取的。我们在程序中设计的报表将输出到这个文件上，即在宽行打印机输出。

在数据部的文件节中(002010行)对此文件进行描述，002030行上“REPORT IS REP”表示报表名为“REP”，其中REPORT是必写字，“REP”是用户自定的报表名。当然也可以写成“REPORT IS A”，这时A就是用户设计的输出报表的名字。

(3) 报表是用数据部的报表节(REPORT SECTION)来定义的，即：程序编制者可以按题目要求设计出各种形式的报表格式。从002200行开始的七行是报表格式的总体的描述。“RD”是表示“报表描述”(我们已学过FD是文件描述，SD是排序文件描述，今RD是Report Description的缩写，意为“报表描述”)。REP是报表名，注意它应和002030中REPORT子句中所用的名字一致。003020 PAGE子句指出“每页的范围指定为66行”，003030指出每页的头从第三行起(即留出二行空白)，003040和003050行指出“报表中打印的细目(detail)行，可以在9—52行之间打印”。如超过52，则不打印在本页上而换页打印。003060行指出，页尾处理从61行以后开始打印。以上这些数字由用户根据需要指定，这样就指定了每页的范围和“天”、“地”的行数。譬如，当页计只需一行(按要求打印在第62行)时，则63—66行总是空白的。

003010行CONTROL子句指定连续打印细目行时遇到什么情况应暂停输出细目行以便进行所需要的“阶段处理”，这叫“控制截断”。我们指定了“FINAL”和两个数据项DE-

PART(系)和CLASS(班)(这二个数据项都已在001130和001140行中定义了)作控制截断项。控制截断指的是对某一数据项的值变化之前和变化之后，暂停细目行的输出，进行必要的处理。例如当按过程部的语句要求读入一个学生记录并生成和打印出它对应的细目行后，在一般情况下继续读入第二个学生记录再打印第二个细目行，……但如果CLASS的值发生了改变，则不连续打印下去，而进行“截断处理”。如果输入文件DATA FILE的第1～5个记录的NUMB值(号码)分别为010101，010102，010103，010105，010201，前四个号码中第三、四个字符(CLASS的值)均为01，而第五个号码中它是02，即CLASS值变了。而CLASS是在CONTROL子句中指定为“控制截断”的项，故连续输出四个学生数据后截断，暂停打印细目行，进行班统计和打印。班统计打印的内容由006080行到007090行中规定。

在一个系的全部班级的学生数据处理完后，下一个学生的学号的前二个数字就会改变。例如为020101，即DEPART的值由01变成02，而DEPART是CONTROL子句中规定的“控制截断”的数据项。因此，在打印020101这个学生数据的细目行之前，要进行“系统计和打印”，“系统计和打印”内容在007100行到009070行中规定。

FINAL表示“最后的控制截断”，即全部学生数据处理完毕后，根据程序编制者的要求作必要的统计打印，其内容在009080行到011030行中规定。FINAL是COBOL规定的保留字，用户不能改变的，但可以不指定FINAL的控制截断，即全部数据处理完后不作任何统计打印。在本例中我们指定FINAL，DEPART和CLASS为控制截断项(当然也可以指定其它项来控制截断，这完全根据需要而定)。应注意这三项的书写次序，FINAL被认为最高级别的控制，DEPART为次高级，CLASS为最低级。每次生成一个细目行之前，先要检查最高级的控制截断是否发生，然后再检查第二级的，然后第三级的……直到最低级别的。如果已发现某个高级别的控制截断已发生，则认为所有比其级别低的控制项都发生了控制截断。例如，当发现DEPART(系)值改变，即DEPART项控制截断发生，则认为CLASS(班)控制截断也发生。FINAL的控制截断只在报表的开头和结尾发生，即仅发生在打印第一个细目行之前和最后一个细目行之后。在CONTROL子句中按控制级别的高低顺序书写(先写级别最高的，最后写级别最低的)。

应该说明，截断后打印的内容是由用户在指定的“控制头栏”或“控制尾栏”中决定的。在本例中，它们是在004030行，004090行，006080行，007100行开始的若干行中指定的。详见下面的叙述。

(4) 从003070行开始共有九个01层号的数据描述体，它们是报表栏描述体。每一个01开头的描述体定义一个打印项目(打印栏)，依次介绍如下：

(i) 第一个报表打印栏在003070行至003120行中描述，REP-HEAD是程序员自定的该栏的名字。TYPE IS REPORT HEADING表示这个报表栏的类型是“报表头”，即根据用户需要，在整个报表的最前面打印所需的某些信息。例如，本例打印一个封面，上面文字为“THE REPORT OF THE STUDENT EXAMINATIONS”(学生考试报表)。003070行中的“REPORT HEADING”是COBOL规定的保留字，不得改变。写上它，在程序运行后，在打印其它内容之前，先打印一个用户设计的“报表头”。003080行中NEXT GROUP IS NEXT PAGE表示打印完这栏后，下一栏(不管下一栏的内容是什么)打印在下一页上，即这个“报表头栏”单独占一页。

第003090行，表示在本页第15行上第40列开始的位置上打印“THE REPORT OF THE

STUDENT EXAMINATIONS" 这样一组字符。第003110行表示在本页50行上第45列开始的位置上打印 "QINGHUA UNIVERSITY，1993.1" 这样一组字符，请对照本例题打印结果看，是很容易理解的。

(ii) 第二个报表栏（003130～004020行）的名字定为 PAGE-HEAD。它的 TYPE 子句指定其类型为 "PAGE HEADING"，即页头，在每页的开头都打印这一栏的内容。只要写上 "TYPE IS PAGE HEADING" 就会自动在每一页上先打一个由用户设计的 "页头"，页头的内容在003140行～004020行中指定，即在 "报表头" 的下一页的第四行第50列开始打印 "QINGHUA UNIVERSITY" 字符。在第六行第45列开始打印 "THE REPORT OF THE STUDENT EXAMINATIONS" 字符。在第七行的110列开始打印 "PAGE"，在同行115列开始，打印页计数。第004020行中有 SOURCE 子句，它表示这个打印项的内容 "来源于" 某一数据项，SOURCE 后面跟一个数据名，此数据项应已被赋值。今在 SOURCE 子句中指定 PAGE COUNTER。PAGE-COUNTER 是系统指定的 "页计数器" 的名字，即每打印完一页，系统会自动往 PAGE-COUNTER 中累加1。如果在打印第二页时，则在页头第七行第110开始会打印出："PAGE 2"。但你所使用的计算机系统是否有此专用的页计数器，可在所用计算机系统 COBOL 说明书。

(iii) 从004030～004070行是 "系表头" 栏。名字取为 DEPART HEAD，"TYPE IS CONTROL HEADING DEPART" 表示此栏的类型是系表头控制。在每遇到一个 DEPART 开头时，在打印本系的其它信息之前先打印此栏内容。004040行中的 "NEXT GROUP PLUS 2" 表示打印完本 "系头栏" 后，下一个栏应从本栏最后一行再隔一行开始。今设计的系表头从哪一行开始打印呢？在004050行上，没写绝对行数，而写 "02 LINE PLUS 2" 表示在上一栏之后隔一行开始打印。今上一栏最后一行是第七行，则加2为9，故 "系表头" 应从第9行开始，它的内容只有一行，打印 "DEPARTMENT" 和 DEPART 的值（即系号）。004070行上 "SOURCE DEPART" 表示在执行打印此行时，将 DEPART 的值打印在70列开始的二列位置上。SOURCE 子句的作用，可理解为与 MOVE 相似，即将 DEPART 的值传送过来，然后打印出来。

(iv) 从004090行开始的是 CLASS-HEAD 栏，从 TYPE 子句中可以看出它是 "班表头" 栏，即每换一个班，则在它的开头打印出它所规定的 "班表头"。"NEXT GROUP PLUS 1" 表示在打印完 "班表头" 栏之后，下一个栏从它的下一行开始打印。004110行为 "02 LINE PLUS 1" 表示 "班表头" 栏的第一行应在原行数基础上再加1，而在004040行中已规定 "下一栏的行数加2"，今再加1，故打印完 "系表头" 栏后，空二行再打印第一个 "班表头"。系表头在第9行，第一个班的班表头应在第12行。班表头的内容是先打印一排 "-"，然后在下一行上打印所需的小标题（统计打印的数据的名称），以便阅读。即打印出："系、班、学号、性别、年龄、课程1到课程5、前三门课分数和、后二门之和、总分、平均分、是否 "通过" 等项的英文字。

(v) 从005100行开始描述 "细目行"，所谓细目行指的是要输出报表的主体，即逐行打印输出的基本数据在本例中为每一个学生数据的行。"TYPE IS DETAIL" 表示此栏内容是 "细目栏"，与前所述规则相同，004100和005110行指定了细目栏的第一行数据应在 "班表头" 行下二行（隔一行）打印。在005120和005130行上有 "GROUP" 这样的字，它是一个 "GROUP INIDICATE" 子句，其中 GROUP 为必写字，INIDICATE 为可选字，可省写。这

个子句的作用是使该初等项(在本例中是 DEPART 或 CLASS)的值仅在控制截断之后，或第一次出现细目栏时才打印。例如一个班有四个学生，都属01系01班，不是在每一个细目行中都把系号和班号都打印出来，而只在第一次出现 DEPART 或 CLASS 时，或发生控制截断时，或虽未发生控制截断而只是换一页时，在第一个细目行上才打印出来。请参看运行打印结果，本栏要求打印的细目行前十项(005120—006010行)的内容都是从读入的记录中获得的。后五项(006020—006060行)的内容是在程序的过程部中经过运算处理得到的，在分析过程部时请注意它们之间的联系。

(vi) 006080行开始的是 CLASS-COUNT 栏，"TYPE IS CONTROL FOOTING CLASS"表示当班号改变时就发生控制截断，打印"班尾"栏。(CONTROL HEADING 为"头"栏，CONTROL FOOTING 为"尾"栏。)在这部分的描述中规定当每个班的细目行打印完之后隔一行打印一排"-"，然后在下一行打印006130到007090行中指定的内容。在006130行上我们定了一个数据项名(即报表栏元素) NUM-OF-CLASS，后面有 SUM 子句。"SUM N"，它表示每次执行细目行的操作处理时，就将 N 的值累加到 NUM-OF-CLASS 中。譬如第一班有四个学生，每打印一个学生的细目行时，N 就加到 NUM-OF-CLASS 一次，在002050行中 N 已被赋予初值1，因此打印四个细目行后，NUM-OF CLASS 的值为4(在每次控制截断后，对"和计数器"如 SUM-OF-CLASS，NUM-OF-FEM-C 等，均重新置零)。在该行的22列开始的三列的位置上打印出 N 的值，同样，每次将 SEX-W 的值(当 SEX 的值为"F"时，即"女性"时，SEX-W 的值为1，见过程部014030行)，累加到 NUM-OF-FEM-C 上，并在27列上打印出全班女同学的人数。006150行只实现将年龄 AGE 累加到 AGE TOTAL-C 上而不打印。在过程部的011190行中将 AGE-TOTAL-C 除以全班人数(NUM OF-CLASS)，得到全班平均年龄 AGE-AVE-C，在34列上打印出来(见006160行)。TOTAL-1-C 到 TOTAL-5-C 分别是五门课全班总分，AVE (1) 到 AVE (5)分别是全班五门课平均分数，在39列，49列，59列，69列，79列上打印出来。在过程部(012050行)中求出全班总平均成绩 AVER-C，在120列上打印出来(007080行)在131列上打印全班有一门以上不合格的人数(007090行)。

(vii) 007100行开始是"系尾"栏，每个系学生数据打印处理完后在同页第53行开始打印一排"-"和全系统计数字。注意，008090行中 SUM 子句中的 NUM-OF-CLASS 就是"班人数"，当打印处理"班尾"栏时，就将它累加到 NUM-OF-DEPART(系人数)上去，然后 NUM-OF-CLASS 重新置零并累计第二班的人数，直到第一个系全部班级的人数都已累加到 NUM-OF-DEPART 上为止。此时发生 DEPART 的控制截断，在22列上打印出全系学生数，同样，打印出全系女生数 NUM-OF-FEM-D。将班年龄总和 AGE-TOTAL-C 累加得全系年龄总和 AGE-TOTAL-D，通过过程部的语句求出系平均年龄 AGE-AVE-D (012090行)，打印在34列上(008120行)。在 TOTAL-1-D 到 TOTAL-5-D 中分别累加全系各班五门课的总分。通过过程部语句 (012100～012140行)求出全系五门课的平均分数，送到 AVE (1)到 AVE(5)中。注意此时是重复使用 AVE(1)到 AVE (5)，以节省内存和少写几行数据描述体。由于班平均分数(每门课的)已打印出来，AVE (1) 到 AVE (5) 原内容不必保存，故可以利用它作为存放系平均分数之用。TOTAL ALL-D 和 AVER-D 分别代表全系总分和全系总平均。

(viii) 009080行开始设计"校统计"，它发生在 FINAL 的控制截断，即打印完最后一个

细目行之后。此时处理控制尾栏的顺序是：先打印最低级的尾栏（“班尾”栏），然后是次低级的尾栏（“系尾”栏），最后是“总尾”栏（“校尾”栏）。由于007110行中有“NEXT GROUP NEXT PAGE”，因此，“校统计”的尾栏换页打印。

(ix) 最后一栏是 REPORT-END，“TYPE IS REPORT FOOTING”表示其类型为“报表尾”，即整个报表的最后部分，它的内容由程序员根据需要设计。011070行中“02 LINE 25 NEXT PAGE”表示报表尾栏是换页打印的，它独占一页或多页。

除以上几种报表栏外，还有“页尾”栏，可以在一页结束时打印所需信息，情况与上述类似（可参考下一个例题），本程序中未用到“页尾”栏。

以上各种报表栏，可以根据需要任选用其中若干个。

注意在报表栏描述中，先后次序应按打印的先后次序书写，即先写报表头栏的描述⇒页头栏⇒系头栏⇒班头栏⇒细目栏⇒班尾栏⇒系尾栏⇒页尾栏⇒报表尾栏。注意，在整个报表中，报表头栏和报表尾栏只能打印一次。

(5) 过程部中的说明部分（DECLARATIVES 开始到 END DECLARATIVES 之间的各行为说明部分），在说明部分中我们根据需要设了三个节（D-1节、D-2节和D-3节）。每个节中第一个为 USE 语句，它用来指定在打印某一报表栏前应执行的一个过程。例如，011180行为 USE BEFORE REPORTING CLASS-COUNT，它表示在生成打印报表栏 CLASS-COUNT（在006080行描述）之前要执行 D-1节中 USE 语句下面的几个语句，即求出班平均年龄、五门课平均分数、班总平均分。这样，在打印“班尾”栏时，所有各项的值都已具备了。

D-2节是表示在执行打印 DEPART-COUNT 栏（007100行描述的）前，先求出系年龄总和、全系各门课平均分数、总平均分数，这些都是打印系尾栏时所要求的数据。如果不在此先计算出来并赋给系尾栏中有关的各项，则不可能打印出我们所需的数据。

同样，D-3节是在执行打印 UNIV-COUNT（“校尾”栏，009080行）前应执行的过程。

这些语句根据实际需要编写，如不需要也可以不设这些节。关于说明语句请参阅本书§13.4。

(6) 过程部中非说明部分，即 S 节，是程序的主要执行部分，由它决定程序执行的步骤。首先，打开 DATA-FILE 和 REPORT-FILE 文件。013120行是“初始化”语句，INITIATE REP 是开始对报表 REP 的处理，如置所有 SUM 计数器为零值，准备生成和打印该报表所有类型的报表栏，将页计数器置为1，行计数器置为零等。

在 PROC 段中读入 DATA-FILE 文件的一个记录，求出前二门课分数之和（SUM1）、后二门课分数之和（SUM2），以及五门课总分 TOTAL，求出该生的平均分数（四舍五入处理）。如有一门以上不合格，则将“NO”送到 PASS 中，将1送到 PASS-W 中，PASS-W 的值累加到 PASS C 中（007090行）可得班不合格人数。如 SEX 的值为“F”（女），将1送到 SEX-W 中，SEX-W 累加到 NUM-OF-FEM-C 中（006140行）。经过以上步骤后，细目栏所需的各数据均已具备。014050行为生成语句，GENERATE DETAIL LINE 语句按005100行开始的细目栏规定的内容生成 DETAIL-LINE 栏（细目栏），并打印出一个细目行，然后再重新转到 PROC 段，再读入和处理第二个记录。在执行第一个 GENERATE 语句时，按下列顺序生成和打印有关报表栏：报表头栏⇒页头栏⇒控制头栏（按级别高低顺序）⇒细目栏。当执行第二个（和以后）GENERATE 语句时，如不发生控制截断，则只生成和打印一个细目栏（DE-

TAIL-LINE)，如果在打印了几个细目行之后发生控制截断，则顺序生成和打印有关的所有控制尾栏，然后从对应于本次控制截断的数据项的控制头栏到最低级的控制头栏顺序生成和打印(如发生 DEPART 的控制截断，则先打印“班尾”栏⇒打印“系尾”栏⇒打印“系头”栏⇒“班头”栏⇒最后再生成打印细目栏)。

在执行 GENERATE 语句时，将执行 SUM 或 SOURCE 语句规定的功能(将 SUM 子句中的数据项的值累加到和计数器中，按 SOURCE 子句要求将数据项的值传送给相应的行列中)。

直到读完全部数据转到 FINISH 段，执行 TERMINATE 语句。TERMINATE 意味最高级控制截断，生成和打印全部控制尾栏，并打印报表尾栏，结束报表功能。因此，在执行一个 TERMINATE 语句以后不能再执行 GENERATE 语句或另一个 TERMINATE 语句，除非重新执行一个 INITIATE 语句。

最后关闭 DATA-FILE 和 REPORE-FILE 文件(TERMINATE 并不关闭文件)。

本程序的流程如图12.3所示。

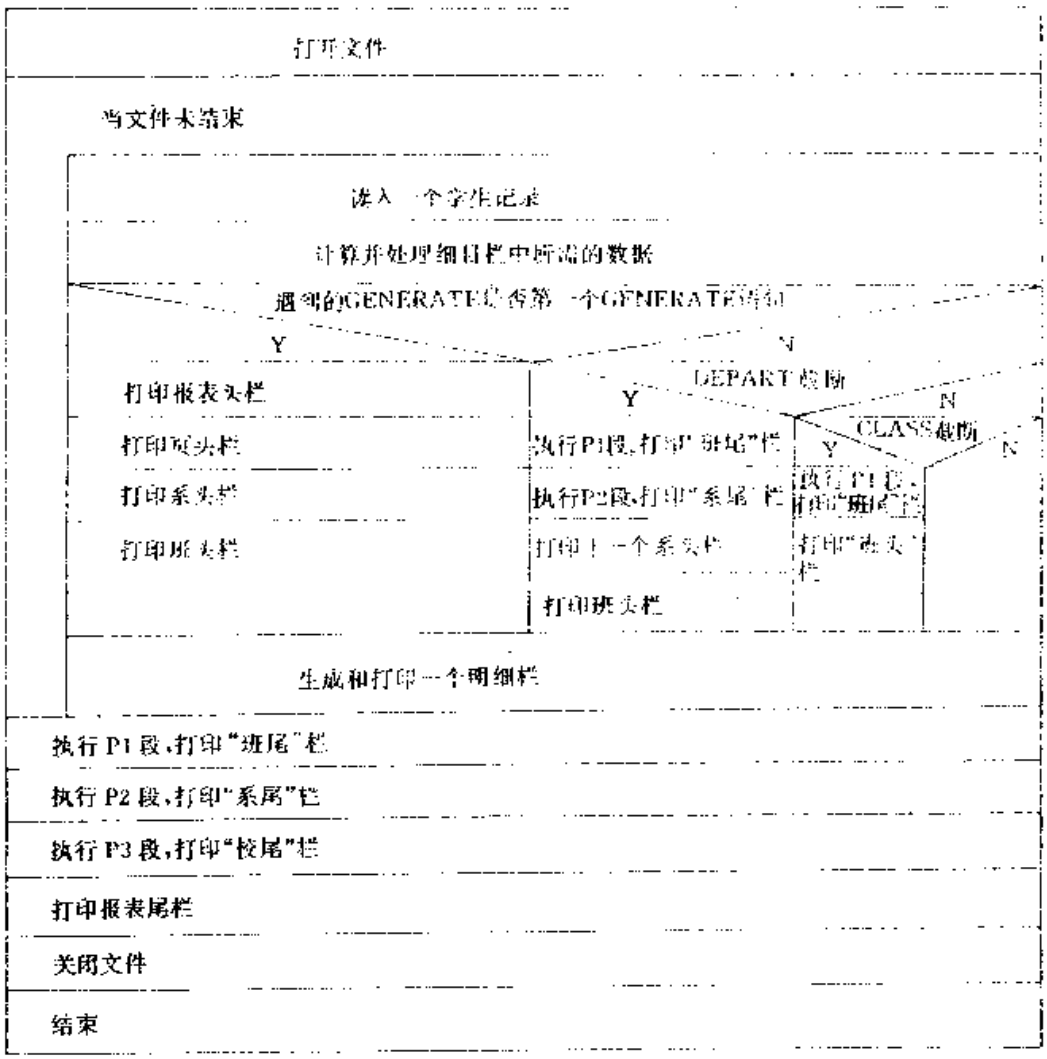



图 12.3

输出报表如下：

# THE REPORT OF THE STUDENT EXAMINATIONS

QINGHUA UNIVERSITY 1993.1

QINGHUA UNIVERSITY

THE REPORT OF THE STUDENT EXAMINATIONS

DEPARTMENT 01 

| DEPART | CLASS | NUMBER | SEX | AGE | COUR-1 | COUR-2 | COUR-3 | COUR-4 | COUR 5 | SUM-1 | SUM-2 | TOTAL | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 01 | 010101 | F | 20 | 90.0 | 88.5 | 76.5 | 73.5 | 93.0 | 255.0 | 166.5 | 421.5 | 84.3 | |
| | | 010102 | | 21 | 92.0 | 76.5 | 60.5 | 53.5 | 83.0 | 229.0 | 136.5 | 365.5 | 73.1 | NO |
| | | 010103 | | 17 | 83.0 | 66.5 | 70.5 | 93.5 | 62.0 | 220.0 | 155.5 | 375.5 | 75.1 | |
| | | 010105 | F | 18 | 61.0 | 93.0 | 70.5 | 63.0 | 68.0 | 224.5 | 131.0 | 355.5 | 71.1 | |
| | | 4 | 2 | 19.0 | 81.50 | 81.10 | 69.50 | 70.80 | 76.50 | | | | 75.90 | 1 |

| DEPART | CLASS | NUMBER | SEX | AGE | COUR-1 | COUR-2 | COUR-3 | COUR-4 | COUR-5 | SUM-1 | SUM-2 | TOTAL | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 02 | 010201 | | 18 | 91.0 | 73.0 | 100.0 | 63.0 | 98.0 | 264.0 | 161.0 | 425.0 | 85.0 | |
| | | 010202 | F | 19 | 75.0 | 79.0 | 87.0 | 53.0 | 99.0 | 211.0 | 132.0 | 393.0 | 78.6 | NO |
| | | 010203 | | 19 | 100.0 | 49.0 | 97.0 | 63.0 | 69.0 | 246.0 | 132.0 | 378.0 | 75.6 | NO |
| | | 010204 | | 21 | 32.0 | 49.0 | 97.0 | 75.0 | 86.0 | 178.0 | 161.0 | 339.0 | 67.8 | NO |
| | | 4 | 1 | 19.2 | 74.50 | 62.50 | 95.20 | 63.50 | 88.00 | | | | 76.70 | 3 |

THE DEPARTMENT COUNT:

| DEPART | NUMBER | SEX | AGE | COUR-1 | COUR-2 | COUR-3 | COUR-4 | COUR-5 | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 8 | 3 | 19.1 | 78.00 | 71.80 | 82.30 | 67.10 | 82.20 | 76.30 | 4 |

* * * * * * * * * * * * * * * * * * * *

QINGHUA UNIVERSITY

THE REPORT OF THE STUDENT EXAMINATIONS

DEPARTMENT 02 

| DEPART | CLASS | NUMBER | SEX | AGE | COUR 1 | COUR-2 | COUR-3 | COUR-4 | COUR-5 | SUM-1 | SUM-2 | TOTAL | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 02 | 01 | 020101 | | 20 | 100.0 | 100.0 | 97.0 | 97.0 | 88.0 | 297.0 | 185.0 | 482.0 | 96.4 | |
| | | 020102 | | 24 | 68.0 | 75.0 | 87.0 | 87.0 | 87.0 | 230.0 | 174.0 | 404.0 | 80.8 | |
| | | 020103 | | 16 | 98.0 | 79.0 | 89.0 | 97.0 | 87.0 | 266.0 | 184.0 | 450.0 | 90.0 | |
| | | 020104 | F | 18 | 99.0 | 56.5 | 89.0 | 97.5 | 87.0 | 241.5 | 184.5 | 429.0 | 85.8 | NO |
| | | 4 | 1 | 19.5 | 91.20 | 77.60 | 90.50 | 94.00 | 87.20 | | | | 88.20 | 1 |

| DEPART | CLASS | NUMBER | SEX | AGE | COUR-1 | COUR-2 | COUR-3 | COUR-4 | COUR-5 | SUM-1 | SUM-2 | TOTAL | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 02 | 02 | 020201 | | 18 | 95.0 | 67.0 | 89.0 | 99.0 | 97.0 | 251.0 | 196.0 | 447.0 | 89.4 | |
| | | 020202 | | 21 | 94.0 | 79.0 | 89.0 | 74.0 | 58.0 | 262.0 | 132.0 | 394.0 | 78.8 | NO |
| | | 020203 | | 23 | 92.0 | 73.0 | 84.0 | 74.0 | 55.0 | 249.0 | 129.0 | 378.0 | 75.6 | NO |
| | | 3 | 0 | 20.6 | 93.60 | 73.00 | 87.30 | 82.30 | 70.00 | | | | 81.20 | 2 |

| DEPART | CLASS | NUMBER | SEX | AGE | COUR-1 | COUR-2 | COUR-3 | COUR-4 | COUR-5 | SUM-1 | SUM-2 | TOTAL | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 02 | 03 | 020301 | F | 20 | 100.0 | 47.0 | 86.0 | 65.0 | 54.0 | 233.0 | 119.0 | 352.0 | 70.4 | NO |
| | | 020302 | | 27 | 87.0 | 77.0 | 60.0 | 65.0 | 57.0 | 224.0 | 122.0 | 346.0 | 69.2 | NO |
| | | 2 | 1 | 23.5 | 93.50 | 62.00 | 73.00 | 65.00 | 55.50 | | | | 69.80 | 2 |

THE DEPARTMENT COUNT:

| DEPART | NUMBER | SEX | AGE | COUR-1 | COUR-2 | COUR-3 | COUR-4 | COUR-5 | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|
| 02 | 9 | 2 | 20.7 | 92.50 | 72.60 | 85.50 | 83.90 | 74.40 | 81.80 | 5 |

* * * * * * * * * * * * * * * * * * * * *

QINGHUA UNIVERSITY

THE REPORT OF THE STUDENT EXAMINATIONS



ALL DEPARTMENT WAS PROCESSED

| NUMBER OF ALL | UNIVERSITY | SEX | AGE | COUR-1 | COUR-2 | COUR-3 | COUR-4 | COUR-5 | AVER | PASS |
|---|---|---|---|---|---|---|---|---|---|---|
| 17 | | 5 | 20.0 | 85.7 | 72.2 | 84.0 | 76.0 | 78.1 | 79.2 | 9 |

REPORT END

WRITTER LI FUX

**【例12.2】** 用COBOL的"报表打印"功能打印学生缺席统计报表。

某学校统计外国留学生住宿请假情况，输入的数据格式为：

| | |
|---|---|
| 学生姓名 | 1～40列(其中1～30列为名，31～40列为姓) |
| 空　格 | 41～42列 |
| 年　级 | 43～44列 |
| 空　格 | 45～46列 |
| 房间号 | 47～49列 |
| 空　格 | 50～51列 |
| 日期(月-日-年) | 52～57列 |
| 空　格 | 58～59列 |
| 缺席标记 | 60列 |
| 空　格 | 61～80列 |

输入的记录是无规律的(不按任何顺序排列的)。

要求按日打印出请假的报表。报表中应按年级、按房间号和名字顺序打印出请假学生名单，并打印出本日请假人数统计，以及到本日为止的累计请假人数，最后打印出本月份请假总人数。报表的格式如图12.4所示(两页合起来是一张报表)。

根据题目要求，采用"报表打印"功能。对输入的各记录应先进行排序处理(按月、日、年级、房号、名字升序排列)，然后顺序打印到报表中。由于要求用英文月名打印，故要将月数转换成英文月名，并按打印要求设计报表节。

程序如下：

```
001000 IDENTIFICATION  DIVISION.
001010 PROGRAM-ID.  EXAM2-2.
001020 ENVIRONMENT  DIVISION.
001030 INPUT-OUTPUT  SECTION.
001040 FILE-CONTROL.
001050     SELECT  INFILE              ASSIGN  TO  FILE1.(输入文件)
001060     SELECT  SORT WORK-FILE ASSIGN  TO  SORTWK.(磁盘文件)
001070     SELECT  REPORTFILE          ASSIGN  TO打印机.
001080 DATA  DIVISION.
001090 FILE  SECTION.
002010 FD  INFILE  LABEL  RECORD  IS  STANDARD.
002020 01  IN-REC  PIC  X(60).
002030 SD  SORT-WORK-FILE  LABEL  RECORD  IS  STANDARD.
002040 01  SORT-REC.
002050   02  STUDENT.
002060       03  NAME-L       PIC  X(30).
002070       03  NAME-F       PIC  X(10).
002080   02  FILLER           PIC  XX.
002090   02  GRADE            PIC  99.
002100   02  FILLER           PIC  XX,
002110   02  ROOM             PIC  999.
003000   02  FILLER           PIC  XX,
```

```
003010  02  MONTH          PIC  99.
003020  02  DAYY           PIC  99.
003030  02  YR             PIC  99.
003040  02  FILLER         PIC  XX.
003050  02  TAL            PIC  9.
003060 FD  REPORTFILE   REPORT IS ABS-REPORT LABEL RECORD OMITTED.
003070 WORKING-STORAGE  SECTION.
003080 77  SAVED-MONTH  PIC  99    VALUE IS  0.
003090 77  CONTINUED    PIC  X(11)  VALUE IS  SPACE.
004000 77  ABSS  PIC  X(8)    VALUE  'ABSENCES'.
004010 77  CA    PIC  X(19)   VALUE  'CUMULATIVE  ABSENCES'.
004020 77  TAL-CTR  PIC  9999.
004025 77  MON  PIC  99.
004030 01  HEAD-1.
004040  02  FILLER      PIC  X(24)  VALUE  SPACE.
004050  02  HEAD-LINE  PIC  X(75)  VALUE  'MONTH        DAY
004060-         'GRADE                    ROOM                  NAME'.
004070  02  FILLER      PIC  X(36)  VALUE  SPACE.
004080 01  MONTH-TABLE.
004090  02  MONTH-1.
005000    03  FILLER  PIC  X(9)  VALUE  'JANUARY '.
005010    03  FILLER  PIC  X(9)  VALUE  'FEBRUARY'.
005020    03  FILLER  PIC  X(9)  VALUE  'MARCH'.
005030    03  FILLER  PIC  X(9)  VALUE  'APRIL'.
005040    03  FILLER  PIC  X(9)  VALUE  'MAY'.
005050    03  FILLER  PIC  X(9)  VALUE  'JUNE'.
005060    03  FILLER  PIC  X(9)  VALUE  'JULY'.
005070    03  FILLER  PIC  X(9)  VALUE  'AUGUST'.
005080    03  FILLER  PIC  X(9)  VALUE  'SEPTEMBER'.
005090    03  FILLER  PIC  X(9)  VALUE  'OCTOBER'.
006000    03  FILLER  PIC  X(9)  VALUE  'NOVEMBER'.
006010    03  FILLER  PIC  X(9)  VALUE  'DECEMBER'
006020    03  FILLER  PIC  X(9)  VALUE  SPACE.
006030  02  MONTH-2  REDEFINES  MONGH-1.
006040    03  MONTHNAME  PIC  X(9)  OCCURS  13  TIMES.
006050 REPORT  SECTION.
006060 RD  ABS-REPORT  CONTROL  ARE  FINAL,MONTH,DAYY,GRADE
006070   PAGE  LIMIT  IS  56  HEADING  2
006080   FIRST  DETAIL  10  LAST  DETAIL  45  FOOTING  54.
006090 01  REPORT-HEAD  TYPE  IS  REPORT  HEADING.
007000   02  LINE  2  COLUMN  57  PIC  X(17)
007010       VALUE  'FED  SCHOOL  SYSTEM'.
```

```
007020  01  PAGE-HEAD  TYPE  IS  PAGE  HEADING.
007030    02  LINE  3  COLUMN    52  PIC  X(26)
007040        VALUE    'STUDENT  ABSENTEISM  REPORT'.
007050    02  LINE  6.
007060      03  COLUMN  56  PIC  X(9)    SOURCE  MONTHNAME (MONTH).
007070      03  COLUMN  66  PIC  X(8)    SOURCE  ABSS.
007080      03  COLUMN  76  PIC  X(11)   SOURCE  CONTINUED.
007090    02  LINE  8.
008010      03  COLUMN  1  PIC  X(132)   SOURCE  HEAD-L.
008020  01  DETAIL-LINE  TYPE  IS  DETAIL  LINE  PLUS  1.
008030    02  COLUMN 24  PIC X(9) GROUP SOURCE MONTHNAME (MONTH).
008050    02  COLUMN  41   PIC  99    GROUP  SOURCE  DAYY.
008070    02  COLUMN  54   PIC  99    GROUP  SOURCE  GRADE.
008080    02  COLUMN  67   PIC  999           SOURCE  ROOM.
008090    02  COLUMN  80   PIC  X(20)        SOURCE  NAME-L.
009000    02  COLUMN  101  PIC  X(10)        SOURCE  NAME-F.
009010  01  CF-1  TYPE  IS  CONTROL  FOOTING  GRADE.
009020    02  LINE  PLUS  2.
009030      03  COLUMN  1  PIC  X(136)  VALUE  SPACE.
009040  01  CF-2  TYPE  IS  CONTROL  FOOTING  DAYY.
009050    02  LINE  PLUS  2.
009060      03  COLUMN  2   PIC  X(12)  VALUE  'ABSENCES  FOR'.
009070      03  COLUMN  24  PIC  Z9  SOURCE  SAVED-MONTH.
009080      03  COLUMN  26  PIC  X  VALUE  '-'.
009090      03  COLUMN  27  PIC  99  SOURCE  DAYY.
010000      03  NO-ABS  COLUMN  49  PIC  999  SUM  TAL.
010010      03  COLUMN  65  PIC  X(19)  SOURCE  CA.
010020      03  COLUMN  85  PIC  999  SUM  TAL.  RESET  ON  FINAL.
010030    02    LINE  PLUS  1  COLUMN  1  PIC  X(132)  VALUE  ALL  '*'.
010040  01  CF-3  TYPE  IS  CONTROL  FOOTING  MONTH
010050      LINE  PLUS  2  NEXT  GROUP  NEXT  PAGE.
010060    02  COLUMN  16  PIC  X(28)  VALUE
              'TOTAL  NUMBER  OF  ABSENCES  FOR'.
010070    02  COLUMN  46  PIC  X(9)  SOURCE
              MONTHNAME  (SAVED-MONTH).
010090    02  COLUMN  57  PIC  XXX  VALUE  'WAS'.
011000    02  TOT  COLUMN  61  PIC  999  SUM  NO-ABS.
011010  01  PAGE-END  TYPE  IS  PAGE  FOOTING  LINE  55.
011020    02  COLUMN  59  PIC  X(12)  VALUE 'REPORT-PAGE'.
011030    02  COLUMN  71  PIC  99  SOURCE  PAGE-COUNTER.
011040  01  REPORT-END  TYPE  REPORT  FOOTING.
011050    02  LINE  PLUS  1  COLUMN  32  PIC  X(13)
011060        VALUE  'END  OF  REPORT'.
```

```
011070 PROCEDURE DIVISION.
011080 DECLARATIVES.
011090 PAGE-HEAD-RTN SECTION.
012000     USE BEFORE REPORTING PAGE-HEAD.
012010 TEST-CONT.
012020     IF MON=SAVED-MONTH MOVE '(CONTINUED)' TO CONTINUED
012030         ELSE MOVE SPACE TO CONTINUED
012040         MOVE MON TO SAVED-MONTH.
012050 END DECLARATIVES.
012060 SORTING SECTION.
012070 SORTER. SORT SORT-WORK-FILE ON ASCENDING KEY
012080           MONTH,DAYY,GRADE,ROOM,STUDENT
012090           USING INFILE
012100           OUTPUT PROCEDURE REPORTER.
013010 END-OF-THE-SORT.
013020 REPORTER SECTION.
013030 INITIATE-REPORT.
013035     OPEN OUTPUT REPORTFILE.
013040     INITIATE ABS-REPORT.
013045 UNWIND-THE-SORT.
013050     RETURN SORT-WORK-FILE AT END
013060         TERMINATE ABS-REPORT CLOSE REPORTFILE.
013070     MOVE MONTH TO MON.
010080     GENERATE DETAIL-LINE GO TO UNWIND-THE-SORT.
```

程序分析：

(1)输入文件的记录 IN-REC 只需用 PIC X(60)描述，因为此记录要送到排序文件记录区进行排序，故只需在排序记录区中作详细的描述即可。

从012070～012100行可看出：排序用自动输入(USING)方式，以 INFILE 作输入文件，先按月由小到大排序，月相同时按日升序排序，然后顺序按年级、房号、姓名的升序排序，输出用输出过程(OUTPUT PROCEDURE)。

“输出过程”指定为“REPORTER”节，在这个节中每次从排序中间文件中收回一个记录 SORT-REC。然后执行一次 GENERATE 语句。由于细目栏 DETAIL-LINE 中各源数据项(即 SOURCE 后的数据项)此时都有值，因此可以输出一个学生的细目行，其中月份已转换用英文。例如，月份为9月时(MONTH 值＝9)，则在008030行中 MONTHNAME(MONTH) 就是 MONTHNAME(9)。在工作单元节中(006030行)表 MONTH-2对表 MONTH-1进行重定义(REDEFINES)，因此 MONTHNAME(9)和 MONTH-1中第九个元素(值为“SEPTEMBER”)同占一段内存，因此 MONTHNAME(9)的值就是“SEPTEMBER”。

打印出来的细目行已按月、日、年级、房号、姓名升序排列好了。

(2)本报表包括：(i)一个报表头栏，在程序和报表中以①指示，打印出的内容为“FED-SCHOOL SYSTEM”(在打印的报表中的第二行上)。

(ii) 页头栏，以②指出，页头栏中的月份也按上法改用英文名字打印出。在第三行上打印出“STUDENT ABSENTEEISM REPORT”，在第六行上打印“SEPTEMBER ABSENCES”。如果第一页打印不完9月的请假学生名单，在第二页继续打印，这时第二页的页头栏有所变化，在第六行上打印出“SEPTEMBER ABSENCES(CONTINUED)”表示是继续打印的页，读者可分析过程部中说明部分中IF语句的作用。SAVED-MONTH是用来存放上一页的月号的，如换页后的月号等于上一页的月号(即MON=SAVED-MONTH)，则在CONTINUED中放入“(CONTINUED)”，并在打印页头时将它打印出来(见007080行)。由于MONTH是控制截断项，不能出现在IF语句中，故先将各记录中的MONTH传送到MON中(013070行)，然后在IF语句中用MON进行比较。

第八行上的内容由HEAD-1提供，见008010行和004030行。

(iii) 细目栏以③表示，因为在006080行中规定“FIRST DETAIL 10”，因此，从第10行开始打印细目栏。由于前三项（月、日、年级）的描述中有“GROUP INDICATE”子句，因此，如果有若干个相同的月名、日子或班级，则只打印第一个月名、日子、班级。

(iv) 控制尾栏CF-1 (009010行)。当年级(GRADE)值改变时，它产生控制截断。在本例中不要求对年级进行统计，故只作简单处理，在打印完一个年级的学生名单后空二行，以④指出。

(v) 控制尾栏CF-2 (009040行)。当日数DAYY变化时产生控制截断，打印出当日的日期和请假人数，以及至当日为止的请假累计数。010000行中有“SUM TAL”，TAL是输入记录中的请假标志，请假则TAL值为1。将TAL值累加到NO-ABS(请假数)中，在第49行上打印出此数字，010020行有“SUM TAL RESET ON FINAL”。SUM TAL的含义与前同，将TAL值累加，但此处要求累加各天的请假人数，如果不写“RESET ON FINAL”则每次出现控制截断后都使和累加器重新置零，即只能求出当日累计数。“RESET ON FINAL”的作用是指定只有发生FINAL截断时才使和计数器值置零，也就是说，一直累加下去，直到最后结束为止。这样打印出来的值就是到当日为止的累计请假人数。如果写“RESET ON MONTH”，表示只有到MONTH控制截断时才使和计数器置为零，这时可累计月请假人数。“RESET ON”后面写控制截断项的名字，但它的控制级别应比此子句所在的控制尾栏的级别高。例如在CF-2栏的描述体中写RESET ON FINAL或RESET ON MONTH都是合法的，但反过来如果在CF-3栏的描述体中写RESET ON DAYY则是不合法的。

控制尾栏CF-2在程序和报表(图12.4)中以⑤指出。

(vi) 控制尾栏CF-3，月统计(010040行起)。以MONTH项作控制截断，在第48行上打印出“TOTAL NUMBER OF ABSENCES FOR”和月名、“WAS”以及本月累计请假数，011000行中的NO-ABS是010000行中由TAL累加得到的每日请假数；然后当一日的细目行打印完后把它累加到TOT中(011000行)，然后NO-ABS就被置为零，再去进行第二天的累加。TOT中是月累加数，当月号变化，TOT又被置为零。图中以⑥指出月统计打印的内容，下一月的报表换一页再打印，见010050行。

(vii) 页尾栏　在55行上打印“REPORT-PAGE-”和页号，图中以⑦指出，页尾按指定只能打印在本页的55行～56行内。

(iix) 报表尾　在下一行打印“END OF REPORT”，图中以⑧指出，打印在56行上，这

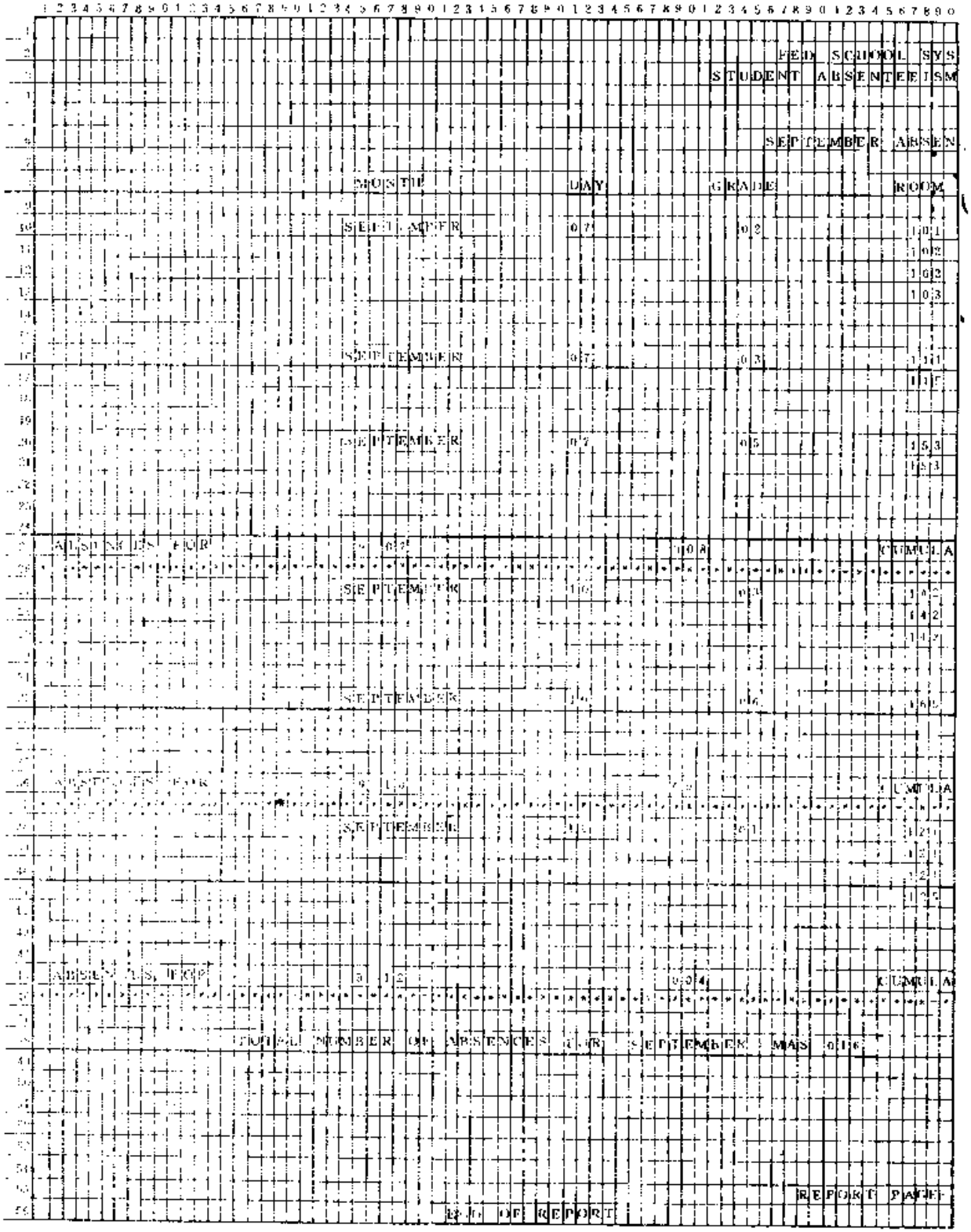

FED SCHOOL SYS
STUDENT ABSENTEEISM
SEPTEMBER ABSEN
POSTIE
DAY
GRADE
ROOM
SEPTEMBER
SEPTEMBER
SEPTEMBER
SEPTEMBER
TOTAL NUMBER OF ABSENCES FOR SEPTEMBER WAS
REPORT PAGE

图 12.5

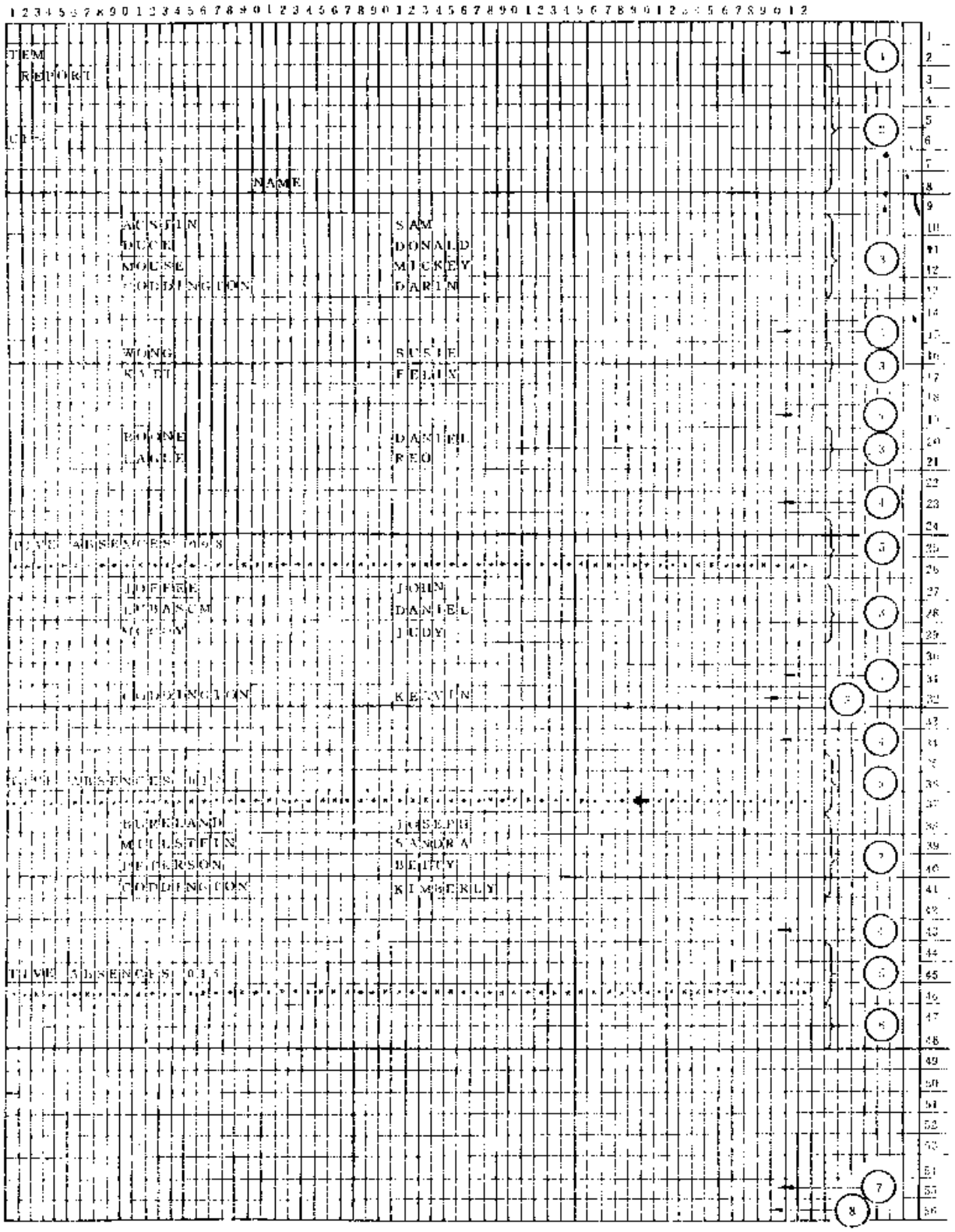

ITEM
REPORT
NAME
ACTION
DUCK
MOUSE
GOOD MORNING MOM
WRONG
KEY
BOONE
LAKE
SAM
DONALD
MICKEY
DARIN
SUSIE
FELIX
DANIEL
REO
JOHN
DANIEL
JUDY
KEVIN
JOSEPH
SANDRA
BETTY
KIMBERLY

图 12.11

是由011050行中的“PLUS”决定的。

读者将程序与打印结果对照分析是不难理解的。

本报表只要求打印报表头、页头栏，没有月、日、年级的头栏，比上例简单些，即在报表描述中没有用到 TYPE IS CONTROL HEADING 的控制头栏。

程序框图见图12.5和图12.6。

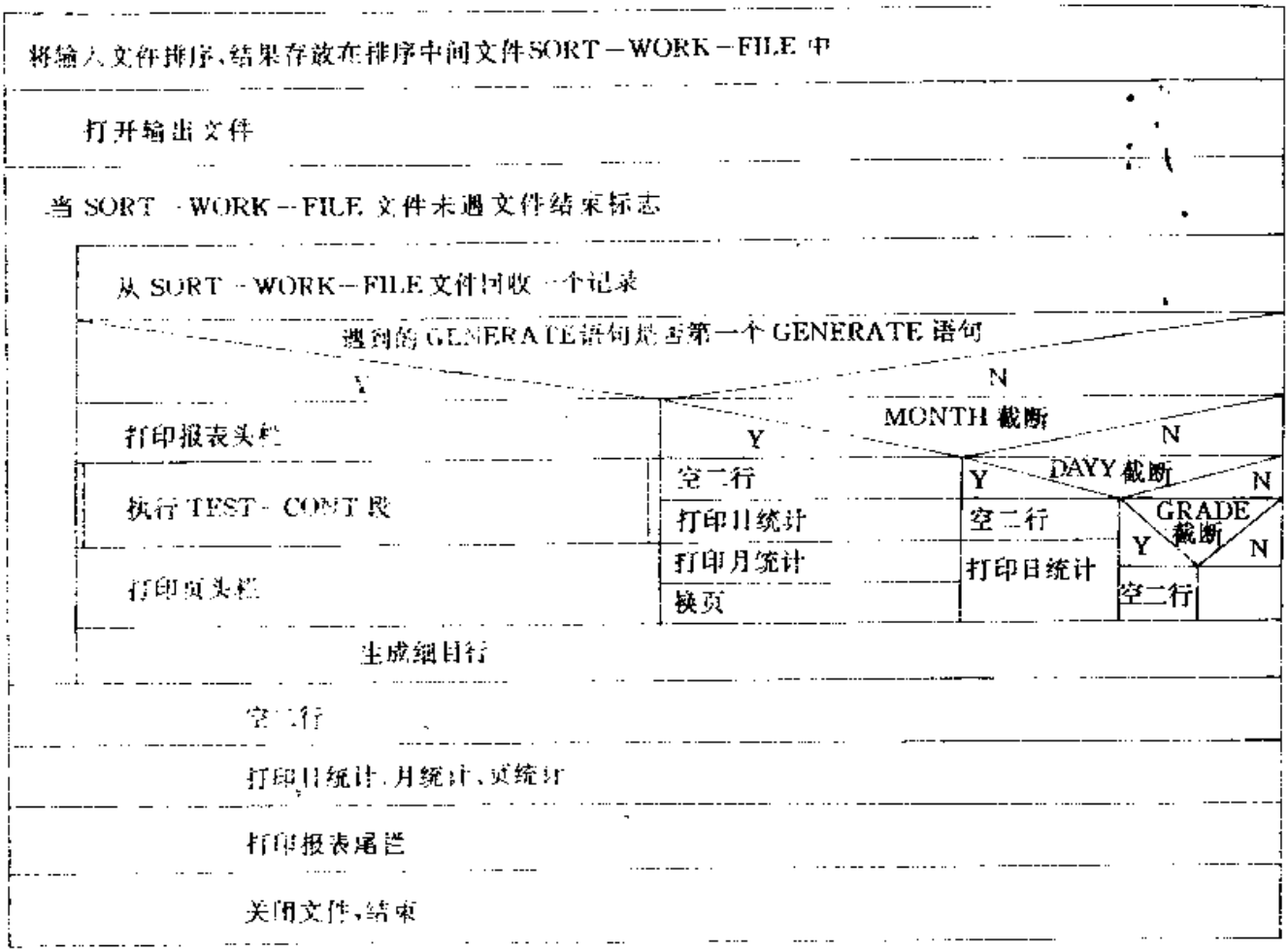



图 12.5

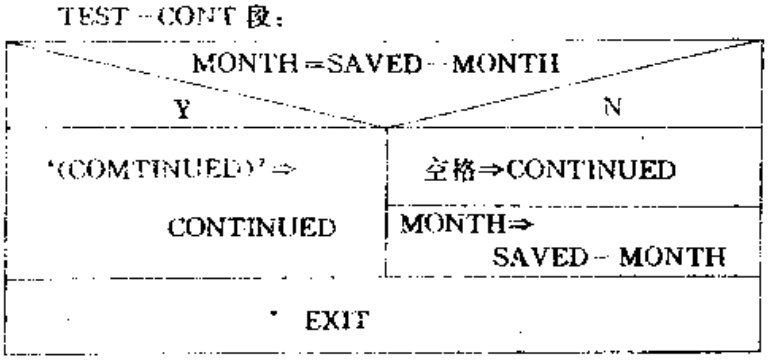



图 12.6

通过以上二例，可以对“报表打印”功能有一初步了解，现归纳以下几点供参考：

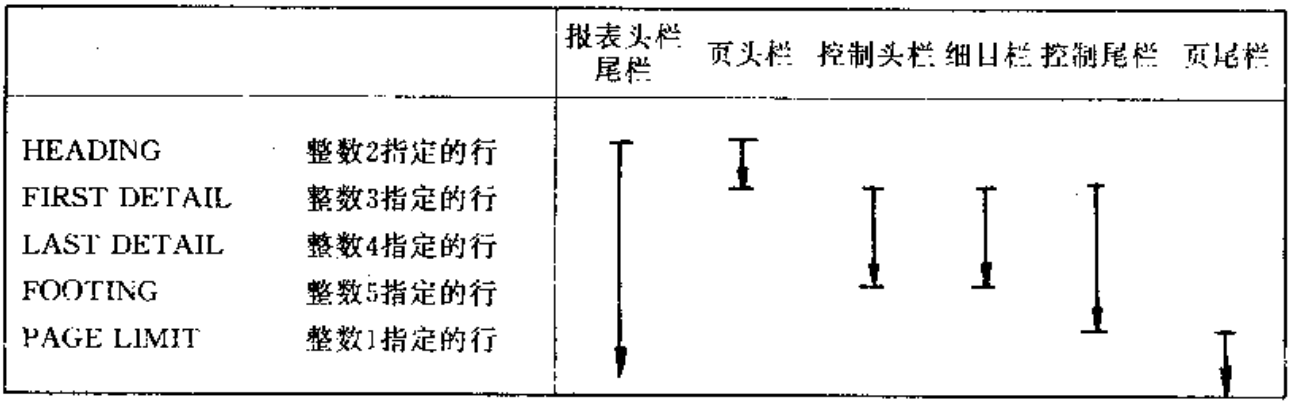
(1) 利用COBOL的“报表打印”功能来输出各种格式的报表(尤其是需要进行各种分

级统计的报表)是十分方便的，它可使过程部大大简化，尤其是减少了大量的 MOVE 语句和 WRITE 语句，使逻辑关系简单明了，思路清楚，大大减少可能出现的编程序中的错误。

利用报表打印功能的程序，主要的工作量是设计数据部的报表节，它甚至会占整个程序的绝大部分篇幅。但是，我们只要掌握它的规律，一层一层地进行设计，并不是很困难的，因为每一种报表栏(如报表头栏、页头栏、控制头栏、控制尾栏、页末栏、报表尾栏)的作用是很明确的，可以根据需要任意选用。我们可以设计好几种典型的报表栏先存在源程序库中，然后在编程序时用 COPY 语句调用它们，并根据具体情况作必要的修改，这样可以减少编写报表节的工作量。

(2) 在报表栏的描述体中，LINE 子句有二种形式：(i) 绝对行数，如02 LINE 36或02 LINE 10 ON NEXT PAGE 等。(ii) 相对行数，在最后打印的行数之上再加一个行数，如02 LINE PLUS 2。可以任选。但在写绝对行数时应正确，例如不应将页头栏的行数写到页尾栏的范围中。

(3) 各报表栏在页面的编排位置如下。




在上面例12.1中，“整数1”为66，“整数2”为3，“整数3”为9，“整数4”为52，“整数5”为61，因此，页头栏应打印在第3行到第8行的范围内，控制头栏打印在第9行到52行间，控制尾栏可从第9行到61行间打印。页尾栏从62行到66行，报表头栏和尾栏可从第3行到66行。

(4) LINE 下面可以包括若干个 COLUMN 子句，如：

```
02 A LINE 50.
   03 B COLUMN 10  PIC  X(8) VALUE 'STUDENT:'.
   03 C COLUMN 20  PIC 9(6) SOURCE NUMB.
```

这时，A 可认为是组合项(报表栏元素组)，B、C 是初等项(报表栏元素项)。如果在本行中只指定一次列数则可合并写为一行：

```
02 A LINE 50 COLUMN 10 PIC X(8) VALUE 'STUDENT:'.
```

同理，如果一个报表栏中只需打印一行，则 TYPE 子句和 LINE 子句可以写在一个描述体中，如上面第二个例题中的008020行中那样，但不能将 TYPE 子句、LINE 子句和 COLUMN 子句写在一个描述项中。如：

```
01 A TYPE IS DETAIL LINE 40 COLUMN 30 PIC X(5) SOURCE B.
```

是不合法的。

只有初等项(带 COLUMN 子句的)的描述体中才能出现 PIC 子句，而且必须有 PIC 子句。如：

03 COLUMN 20 PIC X(10) SOURCE TAL。

初等项的描述除了应写 PIC 子句外，还应具有 SOURCE 子句或 SUM 子句或 VALUE 子句三者之一。即：

{SOURCE IS 标识符1
SUM 标识符2 [标识符3] …
VALUE IS 常量}

不能同时选用其中二个或三个，这三者的作用是给初等项赋予一个确定的数值。

(5) 在 LINE 或 COLUMN 子句所在的描述体中，在层号后面可以写数据名。如例12.1的006130行：

03 NUM-OF-CLASS COLUMN 22 PIC ZZ9 SUM N.

也可以不写数据名，如006160行：

03 COLUMN 34 PIC 99.9 SOURCE AGE-AVE-C.

它只要求在34列上打印出班平均年龄，此外不再使用它了，就不必写名字。而上一个，除了要求在22列上打印 N 的和(班学生总数)外，还要将班总人数保存下来，并将它累加到系总人数中，因此需要有名字才能被引用。在例12.1中的008090行中，调用它累加到 NUM-OF-DEPART 中，如果不给它命名，就无法调用了。

是否写名字(数据名)根据需要而定。当然，一律都写上数据名是个保险的方法(尤其对初学者)，但是多起许多名字，繁琐。当编程序熟悉之后，就可不必都写了。

在报表栏中还可以定义不打印的数据项，如例12.1中006170行，只是要求将 COURS-(1) 累加到 TOTAL-1-C 中，而不打印 TOTAL-1-C 的值。TOTAL-1-C 是在过程部中被调用来求平均分数 AVE(1)的。写006170行的目的是利用 SUM 子句的累加的功能。

注意，数据项可以在文件节中或工作单元节中定义，也可以在报表节中定义。例如这个006170行就是定义 TOTAL-1-C 的描述体，不必再在工作单元节中再对 TOTAL-1-C 定义。不能在数据部中的不同节中对同一数据项重复定义(即描述二次以上)。

(6) 应当强调，报表的生成和打印是由过程部的语句引起的。数据部对报表栏的描述只能提供报表的格式和数据项间的关系(如 SUM，SOURCE 指出的关系)，本身并不产生操作，只有过程部的语句才产生指定的操作(如 GENERATE 语句)。

## 习　题

12.1　数据部中报表节的功能是什么？

12.2　一个逻辑页是如何描述的，报表节中的层次关系是什么？

12.3　把例12.1在你使用的计算机系统上调试通过，并根据自己的需要修改程序，完善改进输出报表。

# 第十三章　程序的编译、运行和提高程序质量的方法

## §13.1　程序的编译、联接和执行

从写好源程序到程序正确执行完毕，一般要经过以下几个步骤。见图 13.1。

(1) 在程序纸上按语法规定写好一个 COBOL 源程序，并反复检查，改正所有已发现的错误。

(2) 将源程序输入计算机，并检查无误。

(3) 对源程序进行编译。将计算机系统提供的 COBOL 编译系统调入内存，由它把源程序转换为机器能执行的机器指令组(即目的程序，或称目的模块)，存放在磁盘上。在编译过程中，还检查源程序中有无语法上的错误，如有，则会打印出错误信息。如果是轻微的错误，系统会自动处理它并继续编译，如是严重错误，则停止编译。

(4) 将主程序和子程序的各个目的模块联接成一个整体。编辑成一个“可执行的程序”。这工作叫“联接编辑”，图 13.2 是示意图。

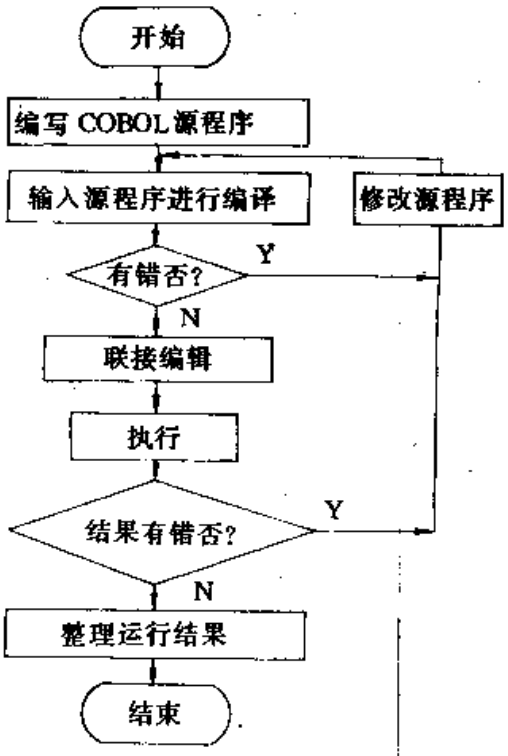



图　13.1

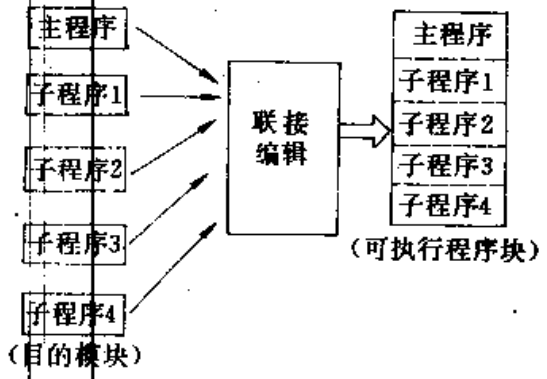



图　13.2

计算机系统在执行联接编辑命令时，先将“联接编辑程序”调入内存，并由它将各目的模块联接起来编辑成一个“可执行的程序”，存放在磁盘上。

如果联接过程中出现错误，会打印出“联接错误”的信息。例如主程序需要调子程序 A，但却找不到子程序 A。

(5)系统从磁盘上调出上面的“可执行的程序”，并执行之。程序中如果有 READ 语句，则在执行到 READ 语句时要求从指定的外部设备上的输入文件中读入数据。程序如果正确，则应得到正确的结果，一般情况下将结果输出到打印机(或磁带、磁盘)。如果执行过程中有错误，计算机又会打印出执行错误的信息。经修改程序后重新编译、联接和执行，直到得到正确的结果为止。

在计算机上进行编译、联接编辑和执行这三个步骤的示意图见图 13.3。

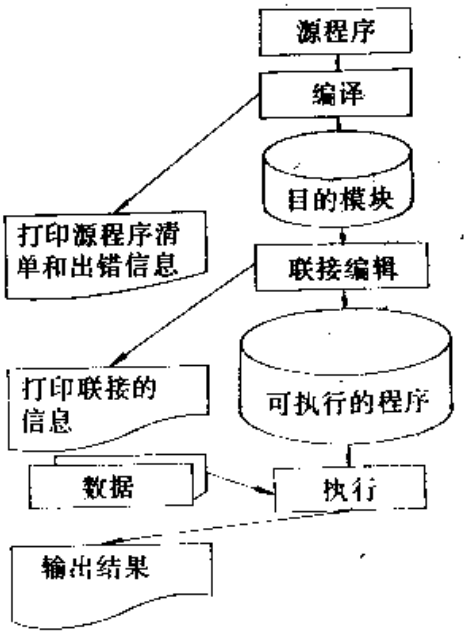



图 13.3

## §13.2 作业的提交

怎样通知计算机进行编译、联接或执行呢？这不是由 COBOL 语言本身来解决的。COBOL 程序设计的任务只是用 COBOL 语言正确地编写出一个 COBOL 源程序。每个计算机系统规定了将 COBOL 程序提交计算机进行编译(compilation)、联接编辑(link editing)和执行(execution)的专用的操作命令。它是属于计算机系统的操作系统的。COBOL 源程序具有通用性(即同一个源程序只需作很小的修改即可以在不同的计算机上运行)，而上述的通知计算机进行编译、联接编辑、执行等操作命令却不是通用的。一种计算机一种格式。因此，要在计算机上运行一个 COBOL 程序，除了要学习 COBOL 语言外，还要知道所用的计算机的有关操作命令。

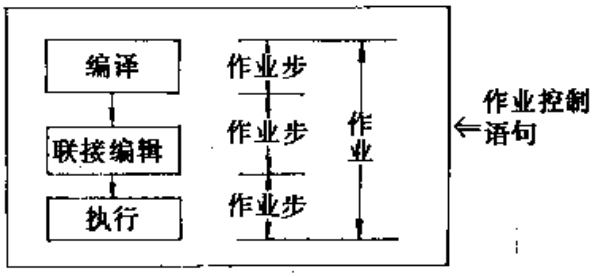



图 13.4

一般中、大型计算机系统有所谓“作业控制语句”(job control statements)，由它来通知计算机执行上述编译、联接编辑和执行等操作的。所谓“作业”(job)是用户提交给计算机一次完成的任务。通常一个作业可以包括若干个步骤，例如：编译、联接编辑和执行。当然也可以是以上步骤之一或一部分。而作业中的每一个步骤称之为“作业步”(job step)。每一个作业控制语句就是通知计算机去完成某一个“作业步”的。要让计算机完成一个作业，需要有若干个作业控制语句。见图 13.4 所示。

为了使读者对这过程有一具体概念。下面我们以两种计算机(中型机和 IBM-PC)的作业提交过程为例来说明之。

### 13.2.1 在中型计算机上运行 COBOL 程序的步骤和作业控制语句

我们以 M-150 计算机为例来说明。其它计算机系统的情况与此可能有所不同，但步骤和方法是差不多的，了解一种之后，可以举一反三。

我们只介绍最简单的情况：

```
//JOB A1                                    (A1 为用户自定的作业名)
//OPTION LINK                               (通知计算机，编译后保留目的模块以便联接)
//EXEC FCOBOL                               (执行编译下面的 COBOL 源程序)
┌──────────────────────┐
│   用户的 COBOL 源程序   │
└──────────────────────┘
CBL                                         (如果不止一个程序块，则用 CBL 行分隔)
┌──────────────────────┐
│   子 程 序 1          │
└──────────────────────┘
CBL
┌──────────────────────┐
│   子 程 序 2          │
└──────────────────────┘
/*                                          (表示源程序结束)
//EXEC LNKEDT                               (执行联接编辑，得到“可执行的程序块”)
//ASSGN SYS010, SYSRDR                      (将 SYS010 分配给读卡机)
//ASSGN SYS012, SYSLST                      (将 SYS012 分配给打印机)
//EXEC                                      (执行“可执行的程序块”)
┌──────────────────────┐
│ 运行程序所需读入的数据记录 │
└──────────────────────┘
/*                                          (需读入的数据)
/EOJ                                        (表示数据到此结束)
                                            (表示作业结束)
```

作业

说明：

(1) 作业控制语句的格式是：第一、二列为两个斜杠，第三列为空白，从第四列开始写作业控制语句的命令。如//EXEC。

(2) 每一个作业控制语句通知计算机完成一个作业步。/* 表示源程序或数据结束，/EOJ表示作业结束。//JOB 表示作业开始。

(3) ASSGN 为分配设备。在 COBOL 源程序中设备部中曾用 SELECT 语句分配设备，如：

```
SELECT CARDFILE ASSIGN TO SYS010-CR.
SELECT OUTFILE ASSIGN TO SYS012-LP.
```

上面的 SYS010-CR 和 SYS012-LP 是程序中用的逻辑设备名，还不是真正的物理设备名。在作业控制语句中，用以下作业控制语句把逻辑设备名和物理设备联系起来。

```
//ASSGN SYS010, SYSRDR
//ASSGN SYS012, SYSLST
```

SYSRDR 和 SYSLST 是 M-150 中指定的读卡片和宽行打印机的名字，这样，就等于将程序中 CARDFILE 文件分配给了读卡机，把 OUTFILE 文件分配给了宽行打印机。

如果用到磁盘、磁带文件，则作业控制语句还要复杂一些，在此不详述。

### 13.2.2 在 IBM PC 机上运行 COBOL 程序的步骤

COBOL 语言的编译和联接等程序，是存储在 COBOL 程序包软盘上的。一般地说，COBOL 程序包软盘有两张，分别标为“COBOL”和“LIBRARY”。

—— “COBOL”盘包括下列文件：

COBOL. EXE

COBOL1.OVR
COBOL2.OVR
COBOL3.OVR
COBOL4.OVR
RUNED.BAT
RUNEC.BAT

——“LIBRARY”盘包括下列文件：

COBOL1.LIB
COBOL2.LIB
COBRUN.EXE
LINK.EXE

（一）COBOL 程序的编辑

COBOL 源程序是一个含有若干个行并以回车换行为结束标志的 ASCII 码的文本文件。可以使用 IBM PC 计算机通用的编辑程序，如用 EDLIN，ED，WS 等来编辑 COBOL 源程序。

（二）COBOL 程序的编译

在操作前，建议复制 COBOL 盘和 LIBRARY 盘。在平时的操作中，使用复制盘，而把原盘保存在安全的地方。

在 LIBRARY 复制盘上，还要复制上 DOS 盘的 COMMAND.COM 文件。

在这些准备工作完成之后，就可以编译 COBOL 源程序了。首先，将 COBOL 盘插在 A 驱动器上，将 COBOL 源程序所在的盘驱动器作为当前驱动器，然后键入命令 A：COBOL，等待片刻后，屏幕将显示头信息和下面的提示信息：

Source filename [.COB]:

然后等待使用者键入被编译的源程序文件名。如果扩展名没有被键入，则 COB 作为缺省值被自动写入。用户回答源程序文件名后，比如说，MYFILE，屏幕继续显示提示信息：

Object file-name [MYFILE.OBJ]

等待使用者键入目标文件的名字。源程序名加上 OBJ 扩展名作为缺省值。如果选用其它目标文件名，扩展名必须为 OBJ。最后显示提示信息：

Source listing [NUT:LS.T]:

等待使用者提供存放编辑后的源程序清单的文件名字。缺省值为不保存源程序清单。

当这些交互式回答全部结束后，编译程序开始工作。如果源程序中存在语法错误，编译程序在屏幕上显示错误信息，同时在清单文件中也列出错误信息。

编译结束后，屏幕上显示编译时所发现错误的个数。如果没有发现错误，则显示：

No Errors or Warnings.

如果编译程序找出了错误，使用者必须在联接之前，找出源程序的错误并改正，重新编译。

（三）COBOL 程序的联接

首先，在 A 驱动器上插入 LIBRARY 的复制盘，接着，键入 A：LINK，屏幕上将显示提示信息：

Object Modules [.OBJ]:

等待使用者输入目标文件的名字。这时不必键入扩展名OBJ。比如，键入MYFILE，屏幕显示提示信息：

RUN FILE [MYFILE.EXE]:

等待使用者输入可执行文件名。缺省值是目标文件名加上扩展名EXE，如果输入其它文件名，扩展名EXE自动加到这个文件名后。屏幕继续显示提示信息：

List File [NUL.MAP]:

等待使用者提供为存放联接程序所产生的打印输出的文件名字。缺省值是不保存这些打印输出。最后一个提示信息为：

Libraries [.LIB]

Libraries关系到IBM COBOL运行用户程序时所必须的各种运行程序，所有这些程序都包含在COBOL1.LIB，COBOL2.LIB和COBRUN.EXE中。

程序库的名字是由目标文件自动提供的。

COBOL1和COBOL2在联接处理时使用，COBRUN在程序运行时使用。

最后在组装了库文件后产生了可运行文件，比如，MYFILE.EXE。

（四）COBOL程序的运行

运行COBOL程序，只须在DOS提示符下键入联接成功后生成的可执行文件的文件名字就可以了。

由于运行COBOL程序时，COBRUN被自动加载，所以，COBRUN.EXE要复制到COBOL程序所在的驱动器磁盘上，或者在A驱动器的磁盘上。

以上是在IBM-PC系列微机上运行COBOL程序的基本步骤。进一步运行及调试，请查阅有关的手册。

## §13.3 程序错误分析和程序的调试

### 13.3.1 语法错误和逻辑错误

COBOL程序的错误有两种：

（一）语法错误(或称编译错误)

写程序时不合COBOL语法规定的，就会在编译过程中被检查出来。例如过程部的语句不从12列开始写而从第8列开始写，或把动词DISPLAY错写成DISPAY等。编译时逐行检查语法。如果是轻微的错误，系统会按一定原则处理它(例如跳过此语句不编译)，而继续编译下去，但仍打印出错信息。用户应修改这些错误，否则程序虽然可能完成编译、联接和执行。但不能保证运行结果是正确的。

每一种计算机系统COBOL都有自己的错误信息表，说明各种错误的含义。

语法错误终究是比较容易查出并改正的，因为翻译时一般的计算机系统会明确指出错误发生在哪一行上。

（二）逻辑错误(或称运行错误)

即使语法没有错误，程序并不一定能正常运行，并得到正确的结果。这是由于程序设计时出现了逻辑上有错误。例如，有一个IF语句：

```
IF PAY<250.0 COMPUTE PAY=PAY * 1.10
```

```
ADD 5.0 TO PAY.
DISPLAY PAY.
```

原意是将工资低于 250 元的加 10%工资，然后再加 50 元补助。250 元以上的不加 10%，只加 50 元补助。见图 13.5。但如果在写 IF 语句时漏了一个句点，成了：

```
IF PAY<250.0 COMPUTE PAY=PAY * 1.10
ADD 50.0 TO PAY.
DISPLAY PAY.
```

则成了图 13.6 所表示那样，对低于 250 元的加 10%，后再加 50 元，而 250 元以上的不加 50 元了。显然得出的结果与原意不同。

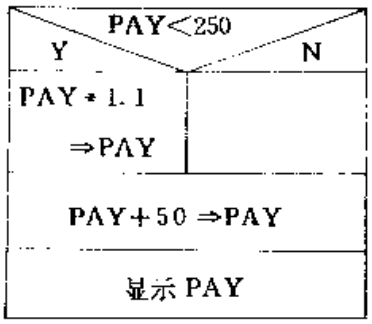



图 13.5

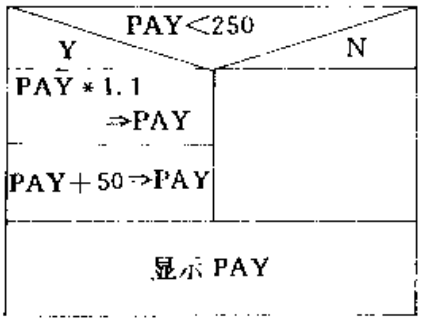



图 13.6

这种错误不出在语法上，语法检查没有问题。程序能执行，但结果不对。这是由于设计程序时有逻辑上的错误，即流程错了。有时是流程图对，但写程序时写错了。有时是框图就画错了，程序当然跟着就错了。这类错误比较难发现，即使发现最后结果不对，往往也不能很快地判断出是哪一段程序或哪一个语句中出了问题。因此，拿到结果后不要认为任务已完成了。应该仔细分析一下结果，看它是否合理。最好在程序通过编译和联接后，先进行“试运行”，即在运行时输入“试验数据”，并在运行后用人工检查结果是否正确。这些“试验数据”应当是数量少，数据简单，便于人工计算。例如原题要计算 1000 个人的平均工资，得到的结果你是很难判断它是否有错。可以先输入三或四个人的工资，工资数都是简单的数字，求出的结果很易用人工核对。试算正确后再输入正式的数据，这样把握大一些。

• 如果发现试算结果不对，则应：①对照流程图逐行检查程序。如程序有错(如上例中少一句点)则修改程序。②如没有发现程序与流程图不符合，则可能流程图画错了，根据题意认真检查流程图，并改正流程图和程序。

### 13.3.2 常见错误举例

下面是初学者在写 COBOL 程序时常常出现的一些错误，在此列出请读者注意。

(1) 在应该出现句点的地方忘写了句点。这类错误非常普遍。如：

```
77  A   PIC  X(16)
```

或

```
DATA DIVISION
```

应记住：每一个部头、节头、段头之后，数据部的每一个数据项的描述体之后，过程

部的每一句子之后都应该有句点和空格。

（2）两个项之间只写了逗号而缺少空格。如：

```
MOVE  3  TO  A, B
```

A 和 B 之间少了空格。在 COBOL 中空格是非常重要的而且是必须存在的分隔符，表示两个成份之间的分隔，不能用逗号或分号代替空格，相反，可以不要逗号而换成空格。

（3）环境部中的 SELECT 子句中的逻辑文件名写错。这是由于有的初学者往往按书上模仿、照抄了书上用的逻辑文件名，结果不能适用于所用的另一型号的计算机系统。因此，编程序时，一定要查清(问清)本系统文件名的规则和用法。

（4）数据部的 FD 文件描述体中，对卡片文件或打印机文件往往漏写 LABEL RECORD IS OMITTED 子句，或错写成 LABEL RECORD IS STANDARD。对不同的文件不会区别应该用 STANDARD 还是 OMITTED。可简单地记住：磁盘、磁带文件用 STANDARD，其它(卡片、打印、纸带、软盘)文件一律用 OMITTED。

（5）在文件节中给数据项赋初值。如：

```
FILE SECTION.
FD DAFILE LABEL RECORD IS STANDARD.
01   DA-REC.
     02   PERSON-NAME PIC X(20).
     02   PERSON-NUM PIC 9(4).
     02   ADDR PIC X(30).
     02   FILLER PIC X(26)VALUE SPACE.
```

文件节中不能赋初值，只能在工作单元节中赋初值。

（6）数据部中描述的数据类型和长度与输入的数据不符合。如描述的是 A PIC 9(6) 而读入的数据却是字符型。这是运行中常出现的错误。有时程序纸上并未写错，而是输入时错了一列而引起的。如：

```
01   A REC.
     02   B PIC 9(4).
     02   C PIC X(20).
     02   D PIC 9(6).
```

在输入数据时，1～4 列应为数字，5～24 列为字符，25～30 列为数字。由于操作员数错了，在 24 列就开始打 6 个数字，第 30 列变成空白，结果就会把字符(空格)读到 D 中去而出错。这种错误不易查出。用户往往只注意检查程序，怎么也查不出错来。如果出错信息告诉你是"数据出错"，应检查输入的数据。

（7）数值型数据项的字段长度不够。如给了：

```
77 K PIC 9(4) V99.
```

由于未充分估计到加减乘除运算后 K 值可能为多大，结果 K 的整数部分超过 4 位而截去高位，导致结果不正确。

（8）把数值编辑型的数据项拿来参加运算，如：

```
77 L PIC $ (4). 99.
77 I PIC 9(4). 99.
```

用了：ADD L TO I，这就错了，编辑数据项一般只供输出时用。

两个编辑项之间也不能传送，如：

MOVE I TO L. 也是错的。

(9) 对编辑型数据项赋予数值初值。

如：77 M PIC 9(3).99 VALUE 0.

(10) 重复使用输出记录区。

```
WRITE OUT-REC AFTER 2.
WRITE OUT-REC AFTER 3.
```

结果在执行第二次 WRITE 语句时，会打印出意料不到的内容。输出记录区在用 WRITE 语句输出一次之后，不能再引用此记录的内容。应再重新给它传送内容。如：WRITE OUT -REC FROM A AFTER 3 是可以的。

(11) 错用了 COBOL 的保留字作用户字(如段名、数据项名)，如用 NUMBER 作数据名，结果打印出错误信息，因为 NUMBER 是 COBOL 的保留字，专有用处。本书后面附有 COBOL 保留字表，读者必要时可查一下，以免用重了。但 COBOL 保留字有三百多个，很难全记住。有一简单办法：在用户字中用连接号，如 DA-REC，PRODUCT-CODE，NAME-CODE 等。保留字中只有很少是包括连接符的。这就是为什么 COBOL 程序中数据名中大量用连接符的原因之一。

(12) 英文字拼错。这特别是在英文程度不大高的同志中容易发生，结果系统不认得此字而不接受，或当作用户自己定义的数据项名来处理。如：

```
MOVE PRODUCT-CODE TO PRODUCT CODE-N
PERFROM PROCESSING.
```

由于把 PERFORM 错写成 PERFROM，编译时就不把它作执行语句处理，而把 PERFROM 当作是用户自己定义的数据名。变成：

```
MOVE PRODUCT-CODE TO PRODUCT-CODE-N, PERFROM, PROCESSING.
```

结果变成将 PRODUCT-CODE 传送给 PRODUCT-CODE-N 和 PERFROM 以及 PROCESSING 三个数据项了。而 PERFROM 又未在数据部中定义，因此，就会出错，打印出“PERFROM 未定义”。

(13) 在用打印机输出时没有注意到在某些计算机系统中，输出记录的第一个字符位置是要被“吃掉”的，它是换行的控制符。结果输出时少了一个字符。如有：

```
01  OUT-REC X(136).
      ⋮
    MOVE 'HOW DO YOU DO'TO OUT-REC.
    WRITE OUT-REC AFTER 2.
```

结果在打印纸上打印出

```
OW DO YOU DO
```

少了第一个字符 H。解决的办法是：

①预留一个空格，即：

```
MOVE  'HOW DO YOU DO'TO OUT-REC.
```

在 HOW 前留一空格，在输出时它被“吃掉”，则输出为：

```
HOW DO YOU DO
```

② 在打印文件的输出区中留一个数据项，占一个字符。如

```
01  OUT REC
  02  FILLER PIC X.
  02  OUT-LINE PIC X(136).
  ⋮
      MOVE 'HOW DO YOU DO'TO OUT-LINE.
      WRITE OUT-REC AFTER 2.
```

则能正确地将 'HOW DO YOU DO' 输出。

(14) READ 后面写了记录名，或 WRITE 后面写了文件名。初学者对此往往易搞错。读者不必死记它，只要理解：外部文件是建立在外部介质上的。因此执行 READ 语句时，是从外部介质上找文件名的。而内存中最大的可存取项是记录，所以 WRITE 语句从内存输出的是记录。至于输到哪个文件上？在数据部中已写明哪个记录属于那个文件。系统会按此关系找到输出文件的。

(15) 数据名不唯一而又未加以限定，如：

```
01  CARD REC.              01  LP-REC.
  02  NUM PIC 9(6).          02  NUM PIC 9(6).
  02  NAME PIC X(20).        02  NAME PIC X(20).
  02  GRADE PIC 9(3).        02  GRADE PIC 9(3).
```

在过程部中用了：

```
ADD GRADE TO TOTAL.
```

而未说明是哪一个记录中的 GRADE.

(16) 程序中用到的某些数据名在数据部中未经定义。这种情况往往发生在：①数据项个数很多而又未经仔细校对；②在调试程序时，临时在过程部中增加了一个数据名，而忘记了在数据部中定义。

(17) 在引用一个已存在的文件时，程序中对该文件的描述与该文件实际状况不符。如已存在一个磁盘文件，每块包含 50 个记录，每个记录有 75 个字节。在程序中要读这个文件，但在数据部的文件描述体中没有说明此文件每块包含多少记录，或指定了与 50 不同的块化因数，或者描述的记录区长度不是 75。这就会在运行中出现致命错误。

(18) 在读写磁盘文件时，在 READ 或 WRITE 语句中把 AT END 和 INVALID KEY 子句用混了。

在顺序读时应一律用 AT END，随机存取时应该用 INVALID KEY。

(19) 在 01 层或 77 层中用了 OCCURS 子句。

(20) 在组合项的数据描述体中用了 PIC 子句。如：

```
01  CA-REC.
  02  STUDENT-ID  PIC X(26).
    03  NAME      PIC X(20).
    03  NUM       PIC 9(6).
  02  ADDR        PIC X(20).
```

(21) 输入或输出时，记录区的长度与设备要求的不匹配，如宽行打印机，有的宽 136 (字符)，有的宽 132，有的宽 120 等等，如果对宽 120 字符的打印机，在输出记录区定义 PIC X(137)，就会出错。

(22)在向一个描述为PIC 9(6)的数据项输入数据时，不写前导零，如只输入␣␣3015，就会出错。因为前两个空格不是数值，应写成：003015。即每一个位置上都必须是数字。

### 13.3.3 程序的调试

对于初学者来说，编写一个COBOL程序，并在上机运行时一次通过而且结果完全正确，是不容易的。尤其是比较复杂的程序，常常是上机几次甚至十几次，反复修改才能通过。即使是有一定经验的程序工作者也难免发生错误，有时在纸上反复检查，自己查不出错，一上机就发现了错误。因此，写出一个程序之后，需要有一个“调试程序”的阶段。所谓“调试”(debug)，是“调整试验”，按英文debug的原意是“捉虫子”，即排除故障，因此，debug亦译作“排错”。调试程序往往会费很多时间，甚至比写一个程序的时间还要多。有的初学者对此估计不足，以为写好程序一上机就能通过，因此，出现错误就急躁、不耐烦，甚至在多次通不过后就没有信心。有的同志在程序一出错时，马上找人帮忙，不愿自己花功夫研究。

其实，调试程序是一个极好的学习机会。有时在书本上、课堂上学不到的东西可以在调试程序中学到。有些语法规定，靠背是记不住的。只有多编程序，多上机调试，出几次错，碰几回壁，就自然记住了，而且印象很深刻。调试程序也是一种学问，有些同志很快就能查出错误，几次就通过了。而有的同志，老找不出错，一个简单的程序十几次都通不过。因此，要不断在调试程序中积累经验，提高技巧。

以下几点供参考：

(1)在第一次上机调试时，可在适当地方加几个WRITE或DISPLAY语句。有的同志写的程序比较长，但直到最后才有WRITE语句。这样如果发生运行错误的话，难以迅速确定出错的区间。如果设几个WRITE或DISPLAY语句，则可以根据打印(显示)出来的信息比较容易判断程序已执行哪一部分，而未执行哪一部分。例如：

```
      ⋮
A.  DISPLAY 'A'.
      ⋮
B.  DISPLAY 'B'.
      ⋮
C.  DISPLAY 'C'.
      ⋮
```

如果程序运行时，已显示出‘B’的信息而没有显示出‘C’，则可以判定出错的区间在B段。这样可以收缩检查范围，较快地找到出错的语句。

在调试时，加一些WRITE，DISPLAY语句对一般初学者是较简单易行的方法，等程序调试通过后，正式运行前再将这些附加的WRITE，DISPLAY语句删掉。

(2)可以利用COBOL的调试语句。有的系统提供了“调试功能”，例如“跟踪”功能，可以将已执行过的段名或段名所在的行号打印出来，这就可以知道程序在哪一段中断运行的。有的系统提供了检查数据项的值是否改变和如何改变的功能，这就可以根据需要确定应检查哪几项的值，然后根据打印出来的结果分析判断。有的系统还可以直接指出执行时发生错误的语句。

每种计算机系统提供的调试功能和使用时的具体规定不完全相同。在调试复杂程序时，灵活地利用调试语句能帮助我们较快地找出问题。读者可参阅所用计算机系统说明书。

(3) 在调试程序时，要注意输入的数据是否合理、正确。不少人在程序出错时只注意查程序本身而不注意检查输入的数据是否正确，而错误往往恰巧发生在数据上（例如提供的数据类型不对，输入数据不够，写数据时最前面多空了一行或在输入数据时多按了一次回车键等…）。

## §13.4 说明部分(DECLARATIVES)和使用语句(USE 语句)的使用

在编制、调试和运行程序中，可以使用 COBOL 的“说明部分”的功能。在过程部中设“说明部分”，使程序设计者可以指定在运行目标程序过程中若出现读写操作的意外情况时，执行“说明部分”中指出的过程。

例如，在过程中有以下形式的 READ 语句和 WRITE 语句：

```
READ CR-FILE AT END PERFORM A.
WRITE OUTPUT-REC INVALID KEY GO TO B.
```

如果在 READ 和 WRITE 语句中不写 AT END 子句或 INVALID KEY 子句而在读写中遇到了 AT END 或 INVALID KEY 条件（例如遇到文件结束记录，或随机写时找不到所指定的磁道），则程序就会中断运行。

我们可以利用“说明部分”指定当上述情况发生时应执行的过程。“说明部分”的位置从过程部头的下面一行开始，即它应该是过程部的最开始的部分。以保留字 DECLARATIVES 开头，以 END DECLARATIVES 结束。在其间写节头，USE 语句和若干个段。USE 语句指出当对指定的文件读写出现错误时应执行它下面的各段过程。如：

```
PROCEDURE DIVISION.                    (过程部头)
DECLARATIVES.                          ("说明部分"头)
A SECTION.
    USE AFTER ERROR PROCEDURE DA-FIE.
    A1. MOVE  'END'  TO END-DATA.
        CLOSE DA-FILE.
    A2. OPEN INPUT DA FILE.
END DECLARATIVES.                      ("说明部分"结束)
```

以上“说明部分”是指定：当对文件 DA-FILE 执行读写时若出现错误，则执行 USE 语句下面的 A1，A2 段（将 'END' 传送给数据项 END-DATA，关闭文件，再打开，以便在主过程中可以从头读文件中记录）。执行完以上过程后，控制返回到引起产生这个操作的地方并继续执行过程部中的后继语句。

USE 语句的一般格式为：

```
USE AFTER STANDARD {EXCEPTION / ERROR} PROCEDURE ON
        {文件名1 [，文件句2] …
         INPUT
         OUTPUT
         I—O}
```

说明：

(1) 除了当读写语句中没有 AT END 子句或 INVALID KEY 子句而在读写操作中具备 AT END 或 INVALID KEY 条件可以引起执行“说明部分”外，在读写时如果遇到硬件错误时(例如磁盘工作不正常)，在自动地执行系统提供的“错误恢复程序”后，也执行“说明部分”的过程。

(2) 可以在“说明部分”中设若干个节(这些节称“说明节”)，每个“说明节”中都要有一个节头，一个 USE 语句和一组过程(由若干段组成)。当一个“说明节”在遇到 END DECLARATIVES 或遇到含有 USE 语句的另一个“说明节”的节名时，就认为本“说明节”结束。

(3)“说明部分”中的过程，只是在读写出错情况下才执行。如果程序正常执行并正常结束，“说明部分”是不起作用的。

(4) USE 语句本身是不执行的，它只指定 USE 过程执行的条件。

(5) EXCEPTION 和 ERROR 两个保留字作用相同，可以任写一个。

(6) 在“说明部分”中不能涉及任何非“说明部分”的过程。例如不能用 GO TO 语句转移到过程部中的“说明部分”以外的执行部分的节或段去。同样的，过程部中非“说明部分”也不能涉及“说明部分”中的过程名。但 PERFORM 语句可以引用“说明部分”的过程。PERFORM 语句所执行的过程必须在同一个“说明节”中。例如 PERFORM A1 TO A3，A1 到 A3 各段都必须属于同一个“说明节”。

(7) 格式中的 INPUT，OUTPUT，I-O 分别指出所指定的文件是：输入文件、输出文件和输入-输出文件。如果写了这些保留字，不必再指出具体的文件名。而认为指定的是全部的输入文件，或全部的输出文件，或全部的 I-O 文件。

利用说明部分，可以使程序适应性更强，尤其在运行大程序时更为显著。

## §13.5 提高程序质量的方法

在学完本书初步掌握 COBOL 语言的基本规则并会编制程序之后，应该逐步提高编程序的技巧和所写程序的质量。

为解决同一个任务，可以编写出不同的程序。但这些程序的质量可能有高低之分。怎样才算一个好的 COBOL 程序？怎样提高程序的质量和编程序的技巧呢？下面介绍的一些是综合了许多使用 COBOL 的人的“经验之谈”。掌握它可以在编程序过程中少走弯路。

(一) 不要写一个庞大的包罗万象的单一主程序。如果程序复杂，可以用若干个子程序来分别完成各部分功能。这样每一个程序单位(主程序或子程序)都比较小，可以分别编译，分别调试。如果某一个语句语法有错，只需修改和重新编译一个子程序，而其它已通过的子程序就不必再编译了。可以提高调试效率，节省计算机的编译时间。

往往把不同的用户都可能用到的部分编成通用子程序，大家都可以使用，节省重复编程序的劳动。

(二) 在一个程序单位内部，最好采用模块式程序结构，也就是将程序组织成不同的功能部分(模块)，用 PERFORM 语句调用这些模块。这是一种较好的组织结构。主模块（控制模块)可以比较短、精炼。如同一个“总调度”。分别调用各处理模块来完成各功能。如：

```
MAIN.
    PERFORM P1.
    PERFORM P2.
    PERFORM P3.
    STOP RUN.
P1. …
P2. …
P3. …
```

如果把段名写成有含义的英文字(或汉字拼音)，则阅读程序更清楚，如同一个“目录”一样：

```
MAIN.
        PERFORM    BEGINNING-PARA GRA PH.    (开始段)
        PERFORM    PROCESSING-PARAGRAPH.     (处理段)
        PERFORM    ENDING PARAGRA PH.        (结束段)
        STOP   RUN.
```

用这种方法设计程序思路清楚，流程图也比较好画。如果出错，可以根据各模块的功能分别查找，也比较容易。

如果程序比较长，每个模块的大小大致以 50 行左右为宜。这只是一般的习惯，并非死规定。

(三) 尽量使程序易于阅读和理解，不要有含混不清的状况。结构化程序设计方法对程序的“清楚性”要比对编程序中的小的技巧的改进看得更重些。一个程序正式调试通过后，可能会提供给成百上千的人阅读，因此一定要考虑到别人阅读程序时的方便。

(1) 尽量用有含义的英文字(或汉字拼音)作段名、节名、数据名，这比用简单的字母(A，B，C…)好，这符合 COBOL“成文自明”的特点。

(2) 写语句时尽量用使人易懂、不会弄错的形式。例如：

```
MULTIPLY K BY L
```

对一些初学者来说，可能会弄不清是 K * L ->K 还是 K * L⇒L？可以用：

```
MULTIPLY K BY L GIVING N
```

形式。即用 GIVING 可选项将二个量的乘积传送给第三个数据项。也可以用 COMPUTE 语句：

```
COMPUTE L =K * L 或
COMPUTE N=K * L
```

显然，COMPUTE 语句要比 ADD，SUBTRACT，MULTIPLY，DIVIDE 语句易于理解，不致含混。COBOL 中用 ADD，SUBTRACT，MULTIPL 和 DIVIDE 语句来表示加、减、乘、除等运算原来是为了增加程序的易读性，即使对数学式子不熟悉的人也能看懂，但对我国这样的非英语国家来说，显然用 COMPUTE 语句比 ADD、SUBTRACT，MULTIPL 和 DIVIDE 语句更易于理解。

当然，ADD 语句用来进行累加是比较方便的。而且运算速度比 COMPUTE 语句稍快一些。

（3）IF 语句要写成易于看懂的形式。如：

```
IF A=B
IF C=D IF E=F PERFORM X
ELSE PEREORM Y.
```

有的初学者可能不易一下子就能看出上面的 ELSE 是和哪个 IF 配对的，会对逻辑流程弄不清楚。最好写成：

```
IF A=B
    IF C=D
        IF   E=F
             PERFORM X
        ELSE PERFORM Y
    ELSE NEXT SENTENCE
ELSE NEXT SENTENCE.
```

一个 IF 与一个 ELSE 配对，逻辑关系清楚。当然 IF 语句的嵌套层最好不要太多。

（4）写程序时尽量写成层次分明的形式。一个语句或一个句子必要时分开几行写，上下两行的开始位置错开几格。如上面的 IF 语句，如果写成：

```
IF A=B IF C=D IF E=F PERFORM X ELSE PERFORM Y.
```

这样的形式就不好看。像上面(3)下面所表示那样，将嵌套的 IF 语句向右错进几格，嵌套关系就一目了然了。

又如：

```
READ IN-FILE AT END MOVE 'END'TO FINISH
    CLOSE IN-FILE STOP RUN.
```

不如写成以下形式：

```
READ IN-FILE
    AT END MOVE 'END'TO FINISH
        CLOSE IN-FILE
        STOP RUN.
```

这样对 READ 句子的范围和 AT END 子句的范围都比较清楚。

（四）尽量少用不必要的英文非必写字。例如：

```
77 PART-NUM PICTURE IS 999999 USAGE IS COMPUTATIONAL.
```

虽然对熟悉英语的人来说，它看起来像一个英文句子，比较好懂。但对我国多数人来说不如

尽量删去不必要的保留字，并用缩写字。如：

77 PART-NUM PIC 9 (6) COMP.

字写多了，总是增加写错或输入出错的可能，因此尽量简化为宜。又如：

IF AMOUNT IS GREATER THAN ZERO DISPLAY AMOUNT.

不如用“>”代替英文字“IS GREATER THAN”，即：

IF AMOUNT > 0 DISPLAY AMOUNT.

（五）要考虑节约计算机时间，特别是减少输入和输出的次数。输入/输出是和外部设备交换信息，比较费时间。对磁盘和磁带的输入/输出可以用“块化”（将若干个记录组成一块，一次读写若干个记录），可以大大缩短输入/输出的时间。指定磁盘文件占一段连续的磁盘空间可以减少磁盘机读写头移动找寻的时间。

用于计算的数值型数据用 USAGE IS COMP 形式，运算所需时间较少。

用 SYNCHRONIZED 子句可以提高操作效率，减少运行时间。

在调试程序时，用 ACCEPT 语句从控制台接收信息或将信息显示在控制台显示屏上有时是可取的。但调试通过后正式运行时，应尽量避免用 ACCEPT 语句从控制台上输入大量数据。

（六）要节约内存，特别对小型、微型计算机尤为重要。不要定义过大的“表”。已用过的数据项的内存单元可以供其它数据项使用。可以多文件共用一个记录区。用 USAGE IS COMP 用法的数据项要比 USAGE IS DISPLAY 用法节省内存（而且运算快），在设计时要估算程序所占内存空间。

（七）要考虑输入数据容易组织；输出打印的结果清晰，意思明白；分页按栏打印，应当不加任何处理即可当作正式文件保存。因此，COBOL 程序输出格式要精心设计。COBOL 有“报表打印”功能。可以使输出格式整齐、多样化（如可设计报表头、页头、页统计…等）。

（八）写程序时要考虑到将来使用和修改程序时的方便。例如，如果每次运行时某些数据是变化的，则不要用赋初值的办法，而在每次运行时临时输入。这样只需改变输入数据而不必须改动程序本身。要考虑到以后程序哪些部分可能要修改，尽量将可能变化的部分减小和集中，譬如，集中在一个子程序中或程序内的一个模块中，其它部分不必改动，便于修改和调试。

（九）不少专家们提出应当尽量减少使用 GO TO 语句。因为 GO TO 语句使程序流程跳来跳去，有时使人看不清楚流程，易出错误。解决这个问题的一个办法是采取“结构化程序设计”的方法。按这种方法写程序，自上而下顺序写下来，执行时也是按顺序执行。流程清楚，不发生跳来跳去的状况。例如有下面一个程序片断：

```
A. IF X NOT > Y GO TO B.
   MOVE X TO Y.
B. IF X NOT >Z GO TO C.
   MOVE X TO Z.
   GO TO D.
C. MOVE Z TO X.
D. DISPLAY X, Y, Z.
```

这段程序虽然不长，但难以一下子就弄清楚，譬如，在什么情况下才转到 C 段和 D 段？

能否从外面转到C段(即C段能否作为GO TO或PERFORM语句的入口)?我们往往在画出流程图后才能搞清楚流程。见图13.7。

我们可以将它改写为:

```
A.   IF X > Y MOVE X TO Y.
     IF X > Z MOVE X TO Z
     ELSE MOVE Z TO X.
     DISPLAY X, Y, Z.
```

作用完全相同,但修改后的这段程序看起来很清楚。见图13.8。可以看出整个A段是一个整体。原来的C段(即MOVE Z TO X)只是当X>Z为"假"时才会执行,它不能作为外边转入的入口。原来的D段(DISPLAY语句)是一定执行的。

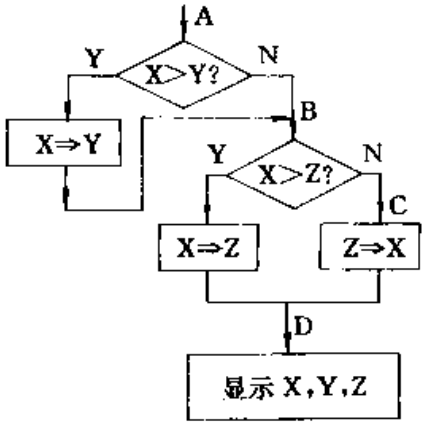



图 13.7

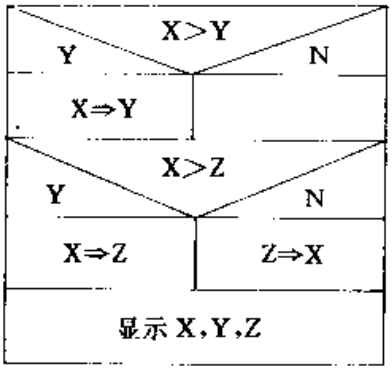



图 13.8

显然,这样的程序只要顺序看下去,流程很清楚,不会跳来跳去,不会产生许多分支。因此,从易于阅读考虑,即为了程序的"清楚性",不少人主张用这种没有GO TO语句的结构化程序设计。但并不是说在程序中都不要用GO TO,GO TO有时还是很有用的。只是说,不要多用GO TO语句转来转去,使流程不清楚。

程序中应当由本书上册已介绍过的三种基本结构(顺序结构、选择结构、循环结构)组成。

## §13.6 COBOL-85对COBOL-74的发展

继ANSI COBOL-74后的新版本ANSI COBOL-85的主要特点是,通过语句的格式的变化及增加新的语句来加强COBOL的结构化程序设计特点,下面分别加以概述。

### 13.6.1 在某些语句中增加END结尾字(END-)

这些增加了END-结尾字的语句是:

| | |
|---|---|
| ADD----------END-ADD | READ----------END-READ |
| CALL--------END-CALL | REWRITE-------END-REWRITE |
| COMPUTE----END-COMPUTE | SEARCH--------END-SEARCH |
| DELETE------END-DELETE | STRING---------END-STRING |
| DIVIDE-------END-DIVIDE | UNSTRING-----END-UNSTRING |

IF---------- -- END-IF　　SUBTRACT-----END-SUBTRACT

MULTIPLY---END-MULTIPLY　　WRITE-------- END-WRITE

PERFORM-- -- END-PERFORM

### 13.6.2 PERFORM 语句的改进

ANST COBOL-74的 PERFORM 语句有两个不足：

(1) PERFORM 语句所要执行的操作总是置于另外的段和节中，这种形式相当于其它语言的子程序调用，对实现模块化是方便的，但当用 PERFORM 语句实现循环结构时，循环体置于另一处特别是循环体规模不大时，削弱了程序易读性，而在 FORTRAN 或其它语言中，循环语句和循环体都放在一处，加强了程序的易读性，例如：读5个记录并将其输出的 FORTRAN 77程序为：

```
      ⋮
     DO 10, I= 1, 5
            READ   (1, *)   A1, A2, A3, A4, A5(文件记录中各项)
            WRITE  (2, *)   A1, A2, A3, A4, A5
10   CONTINUE
```

(2) 我们已在过程部 PERFORM 语句使用的介绍中看到，PERFORM…UNTIL 语句格式既不完全对应 DO WHILE 结构，又不完全对应 DO UNTIL 结构，用 COBOL 的 PERFORM 语句去实现 DO WHILE 或 DO UNTIL 时，需要做某些补充处理。ANSI COBOL-85 针对这些缺点进行了如下改进。

(1) 针对 ANSI COBOL 74的 PERFORM 第一个缺点，新的语句改变了原语句格式。相应的新语句格式为：

**格式一**

PERFORM [过程名1 [{THRU / THROUGH} 过程名2]] {标识符1 / 常量1} TIMES

[语句序列 END-PERFORM]

**格式二**

PERFORM [过程名1 [{THRU / THROUGH} 过程名2]]

[WITH TEST {BEFORE / AFTER}]

VARYING {标识符2 / 位标名1} FROM {标识符3 / 位标名2 / 常量2} BY {标识符4 / 常量3} UNTIL 条件1

[AFTER {标识符5 / 位标名3} FROM {标识符6 / 位标名4 / 常量3} BY {标识符7 / 常量4} UNTIL 条件2]

[语句序列　　END-PERFORM]

我们看到两点改变：第一，被执行过程部分变为可选部分(放在方括号内)。第二，增加了可选项“语句序列 END-PERFORM”，即增加内置语句部分，通过下例可看到新语句用法：

```
PERFORM
    VARYING ITEMC FROM 1 BY 2.
            UNTIL ITEMC>7
                    MOVE CHARA (ITEMC) TO CHARB (ITEMC)
                    MOVE CHARA (ITEMC) TO CHARB (ITEMC+3)
    END-PERFORM.
```

(2) 针对 ANSI COBOL-74 PERFORM 中第二点缺陷，在 ANSI COBOL-85语句格式中增加了 TEST $\left\{\begin{matrix}\text{BEFORE}\\\text{AFTER}\end{matrix}\right\}$ 选项，从而能够用 PERFORM 语句直接实现 DO UNTIL 控制结构。实现 DO UNTIL 仍要使条件取反才行，如下面右边的伪代码和左边 COBOL PERFORM 语句功能是等效的。

```
PERFORM WITH TEST AFTER UNTIL 条件1
        语 句 序 列
END-PERFORM.
PERFORM WITH BEFORE UNTIL 条件2
        语 句 序 列
END-PERFORM.
```

```
DO UNTIL 条件1
    语句序列
ENDDO
DO WHILE 条件2
    语句序列
ENDDO
```

### 13.6.3 IF 语句的改进

在 IF-THEN-ELSE 语句中增加 END-IF 选择以代替原来语句中的“句号”，使得 IF 语句嵌套很容易，并且层次清楚，易读性强，例如对应图13.9的 COBOL 程序段有下列三种情况。

```
IF  p          IF  p          IF  p
  IF  q          IF  q          IF  q
    F1             F1             F1
  ELSE           F3           ELSE
    F2           ELSE             F2
    F3             F2           END-IF
ELSE               F3         ELSE
    F4.      ELSE                 F4
                   F4.        END-IF.
```

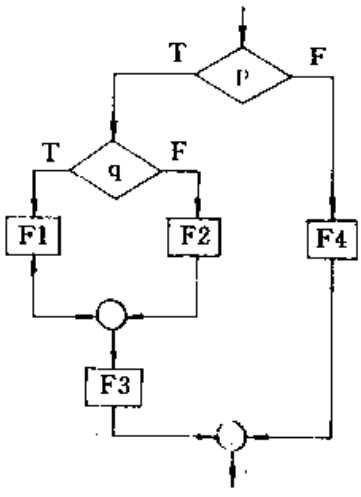



图 13.9

上面左边的程序段是用 ANSI COBOL-74直接按流程图写的，其结果显然是错误的。中间的程序段是用 ANSI COBOL-74按照流程图并经过处理写出的，其结果是正确的。右边的程序段是用 ANSI COBOL-85按照流程图直接写出的，其结果也是正确的，由此可见增加了 END-IF 选项给程序设计带来了多么大的方便。

COBOL-85的 IF 语句有明确的范围。

它的一般形式为：

```
IF    条件    THEN    语句1
      [ELSE    语句2]
END-IF
```

IF 和 END-IF 形成一个 IF 语句的“范围”。IF 语句可以嵌套，即可以在一个 IF 语句中出现多对 IF 和 END-IF，同一层中的 IF 和 END-IF 相对应。

如果在 COBOL1974中有以下 IF 语句：

```
IF  A  B
    IF C=D ADD C TO D
    ELSE
      ADD E TO F.
```

这样写，ELSE 是和哪一个 IF 对应的，往往容易搞混淆，可以用 SPCOBOL 改写为：

```
IF  A=B
 IF  C=D
     ADD C TO D
 ELSE
       CONTINUE
 END-IF
ELSE
        ADD E TO F
END-IF.
```

这样就清楚多了，每个 IF 语句和同一层的 END-IF 相对应。这样的 IF 语句称作“带范围的 IF 语句”，或简称“范围 IF”。

注意：(1) 格式中的“语句1”和“语句2”必须是无条件语句(强制语句)或带范围的语句(“范围 IF”，或下面介绍的“范围 PERFORM”或“范围 EVALUATE”语句)。

如下面的写法不合法：

```
IF  A=B
    IF    C=D
          ADD C TO D
ELSE ADD E TO F
END-IF
```

因为嵌套在内层的 IF 语句不带 END-IF，它的作用范围没有指定。在 IF 语句的嵌套中，不能用不带 END-IF 的“非范围 IF 语句”。

(2) 可以不写 ELSE 可选项，此时当条件为假时，就转去 END-IF 处。

(3) IF 语句的作用范围以遇到 END-IF 或遇到句点和空格即停止。如出现句点和空格，表示一个句子结束。例如：

```
IF  A=B
    IF    C=D
          ADD C TO D
```

```
ELSE
        CONTINUE
    END-IF.
ELSE
    ADD E TO F
END-IF.
```

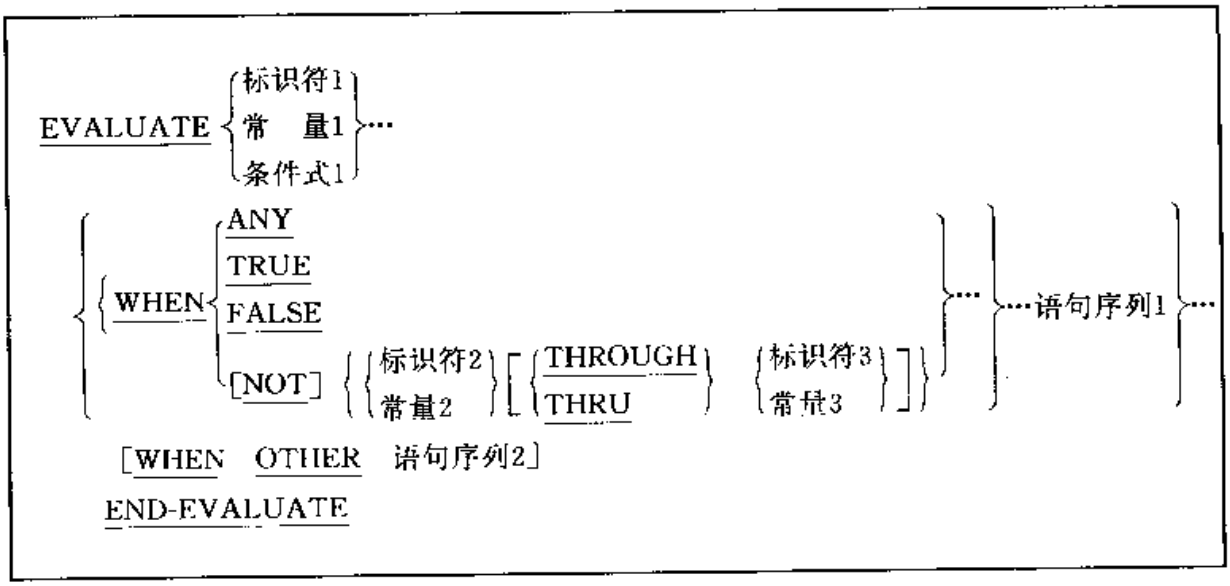
在第一个 END-IF 后面多加了一个句点。此时，系统就认为整个 IF 句子结束，而将它下面一行的 ELSE 作为下一句子的开始，这时就出现编译错误。因此应记住：在嵌套的第一个 IF 和最后一个 DEN-IF 之间不能出现句点。

### 13.6.4 增加了多分支选择语句(EVALUATE 语句)

ANSI COBOL-74没有多分支选择语句，因此遇到这种情况，只能用 IF-THEN-ELSE 或 GO TO DEPENDING ON…语句来模仿，因而使程序繁琐并且容易出错。

COBOL 85增加了多分支选择语句。

其一般格式为：




将“判别主体”(即标识符1，常量1，条件式1)的值和在 WHEN 子句中指定的“判别对象”(ANY，TRUE，FALSE，标识符2，常量2等…)作比较，以判定它们是否相等，若相等，则执行语句序列1，否则执行语句序列2(如果没有 WHEN OTHER 选择项，则不执行语句序列2，而直接从 END-EVALUATE 出口)。

例如：

```
EVALUATE A
        WHEN  3  DISPLAY  A
        WHEN  4  DIVIDE  2  INTO  A  DISPLAY  A
END-EVALUATE.
```

当 A＝3时，显示 A，当 A＝4时，A 先被2除，然后显示 A 的新值。

说明：(1)“判别主体”的个数可以不止一个，但此时“判别对象”的个数应与之相同。执行时，将“判别主体”和“判别对象”一一对应检查，只有当它们全部对应相等时才执行语句序列1。如：

```
EVALUATE A, B, C
    WHEN  1, 2, 3    DISPLAY A, B, C
END-EVALUATE.
```

当A=1,B=2,C=3时，显示A,B,C值。只要三者之中有一个不相等，都不执行DISPLAY语句。

(2) TRUE和FALSE表示“真”和“假”，它和“判别主体”中条件式或标识符(其值只能为“真”或’假”之一)相对应。条件式可以是简单的条件也可以是复合的条件。

举例：假如已经建立了一个“表”，其中有100个元素，每个表元素的内容为姓名，性别、年龄。要求将其中姓名为“LI FUN”，性别为“女”(以“F”表示)，年龄为45岁的打印出来。

可以在过程部中写出以下的程序片断。

```
MOVE 0 TO I.
PERFORM TEST BEFORE
          UNTIL  I  NOT < 100
  ADD  1  TO  I
 EVALUATE   NAME (I), SEX (I), AGE (I)
       WHEN 'LI FUN','F',   45
            DISPLAY NAME (I), SEX (I), AGE (I)
 END-EVALUATE
END-PERFORM
```

### 13.6.5 其它方面的改进

除了上述过程部分增加新的语句外，新的版本还作了其它方面的改进，如使环境部配置节变成可选，并增加了记录定界符选项，重定义数据项允许与被重定义数据的长度不同，可以从一个字符串中提取子串等，为了使读者对新版本有一个全面概括的了解，附录中列出了新版本ANSI COBOL 85语句通用格式汇集。对于想更详细了解新版本内容的读者，请阅读有关厂家ANSI COBOL-85的参考手册。

## §13.7 关于汉字COBOL

COBOL主要用于数据处理，打印报表是其主要功能之一。但在COBOL程序中使用的是英文。所用的字符也是英文字母加上数字和一些专用字符，不能处理中文。也就是说，打印出的报表除数字和专用字符外，只能用英文或汉字拼音来表示某一含义。如要打印“产品名”,“价格”,“数量”等表头，只能用英文字PRODUCT-NAME,UNIT-PRICE,QUANTITY等来表示(或用汉语拼音，或用用户自己规定的符号)。对我国广大用户来说，这是很不方便的。人们希望能打印出中文字，如果能打印出如下页开头的报表是会使用户满意的。

这样，用COBOL打印出来的报表可以不经任何加工就作为档案或资料保存。用中文打印通知单(收费单、成绩单…)更为用户所欢迎。因此在COBOL中使用汉字是我国广大用户的一项迫切要求。为此，需要使用汉字终端、汉字打印机和处理汉字的软件。

| 产品号 | 产品名 | 单价 | 数量 | 总数额 |
|---|---|---|---|---|
| 0001 | 帽子 | 5.60 | 5 | 28.00 |
| 0002 | 钢笔 | 1.70 | 10 | 17.00 |
| 0003 | 铅笔 | 0.06 | 120 | 7.20 |
| 0006 | 练习本 | 0.12 | 50 | 6.00 |
| 0010 | 袜子 | 3.10 | 6 | 18.60 |

目前国内外已研制成功多种汉字终端，它们可以接收汉字代码和输出汉字。在输入时，可以采用各种不同的汉字输入方法。

目前使用的汉字COBOL，指的是仍然用COBOL语言编程序，只是在“非数值常量”中可以使用汉字。例如：一个输出记录的描述为：

```
WORKING-STORAGE SECTION.
01  OUT REC.
      02  FILLER PIC X(10).
      02  FILLER PIC X(6)     VALUE IS '产品号'.
      02  FILLER PIC X(6)     VALUE IS SPACE.
      02  FILLER PIC X(6)     VALUE IS '产品名'.
      02  FILLER PIC X(6)     VALUE IS SPACE.
      02  FILLER PIC X(4)     VALUE IS '单价'.
      02  FILLER PIC X(6)     VALUE IS SPACE.
      02  FILLER PIC X(6)     VALUE IS '总款额'.
```

如果在过程部中有以下输出语句：

```
DISPLAY OUT-REC.
```

则会显示或打印出本页开始时所要求打印的中文表头。

用汉字终端输入信息时，可以混合使用英文和汉字。在打印源程序清单时，可以打印出包括汉字的COBOL源程序（如本页中部列出的程序片断）。在输出运行结果时，既可以输出所需要的汉字，又可输出各种数字和字符。

详细的使用说明可参阅各计算机系统的汉字处理系统。相信汉字处理系统的出现将使COBOL得到更广泛的应用。它能打印出任意的中文信息，甚至一篇中文文章。用中文打印的报表绝大部分中国人都能看懂，为在我国实现计算机管理提供了很大的方便。

## §13.8 程序说明书的书写要求

一个有价值的COBOL程序一经正式通过，就可以而且应该提供出来供大家使用。COBOL语言主要用于数据处理。因此一个程序往往可以用很长时期，只是每次输入不同的数据而已。例如：工资统计、人事管理、产品库存统计、户籍管理、图书检索、销售统计、银行帐目来往等都是这样。因此，要做好程序的交流工作。

为了使别人能看懂和使用自己编的程序，一个正式的程序说明书应当包括以下几部分：

（一）题目（程序标题）

（二）题目要求
（三）题意分析和解题方法
（四）程序中所用的数据名的含义
（五）流程图
（六）程序清单
（七）输入数据和输出结果(可以用试验数据作说明)
（八）使用程序的注意事项

# 附　录

## 附录 I　COBOL 语言的功能模块

为了便于不同的厂家设计和实现 COBOL 语言的编译程序。ANSI COBOL 1974 将 COBOL 语言按数据处理的主要内容，划分为十二个功能模块。或者说，COBOL 由十二个功能模块组成。每个功能模块又分为一级和二级，二级的功能强于一级，不同的厂家可以根据不同规模的计算机系统，从中选取若干个功能块构成自己的 COBOL 编译系统。

下表为 COBOL 语言功能块和等级划分的情况，其中 0 级表示“空”，即可以没有此种功能。由下表可见，最小的 COBOL 至少应包括一级核心，一级表处理和一级顺序存取。最大的 COBOL 则可包括所有十二个功能块，并取最高的级别。因此，用户应了解所用的计算机系统配置的 COBOL 的规模，以便正确地使用它，或者根据自己业务的需要去选择能满足要求的 COBOL 编译系统。

| 功能块 | 核心 | 表处理 | 文件处理 | | | 排序和合并 | 报表打印 | 程序分段 | 源程序库 | 程序调试 | 程序联接 | 程序间通信 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 顺序文件 | 相对文件 | 索引文件 | | | | | | | |
| 等级 | 二级 | 二级 | 二级 | 二级 | 二级 | 二级 | 一级 | 二级 | 二级 | 二级 | 二级 | 二级 |
| | | | | 一级 | 一级 | 一级 | | 一级 | 一级 | 一级 | 一级 | 一级 |
| | 一级 | 一级 | 一级 | 0 级 | 0 级 | 0 级 | 0 级 | 0 级 | 0 级 | 0 级 | 0 级 | 0 级 |

核心：　COBOL 语言中的最基本的部分。

表处理：　对组织成表(Table)的数据进行处理的方法。

文件处理：　对组织成文件的数据进行处理的方法。ANSI COBOL 1974 有三种文件组织(顺序文件、相对文件、索引文件)，而 ANSI COBOL 1968 有顺序文件和随机文件。

排序和合并　(即分类)：根据记录关键字将文件中各记录按此关键字的值大小排序。

报表打印：　制定了一些简便的方法用来打印某些最常见的报表格式。

程序分段：　即程序覆盖，如程序太大，可用分段技术，将目标程序一段一段调入内存，将原在内存中的部分程序段覆盖。这样就可以运行任何大的程序。

源程序库：　将需用的 COBOL 源程序的各种片断，放入源程序库中。不同用户可根据需要，调出源程序库中各部分内容装到自己的程序中，可节省编程序时间。

程序调试：　COBOL 提供一些用来调试源程序的语句。一旦调试通过，它们不再起作用。

程序联接：　COBOL 程序可调用其它 COBOL 程序或用其它语言写的程序(子程序)，将它

们联接起来。

程序间通信：指计算机网络中用户与计算机通信的功能。这是为了计算机网络的用户使用COBOL语言方便而设计的。

## 附录Ⅱ　关于COBOL语言格式的说明

COBOL的语法规则是通过语言格式表来描述的。程序设计者必须严格按照语言格式表的规定来编写程序。在本书附录Ⅲ和附录Ⅵ中给出了ANSI COBOL 1974和ANSI COBOL 1985的语言格式表。

为了书写，阅读和使用的方便，在本书中(以及在大多数COBOL书中)作以下的约定：

(1) 在语言格式中，大写的英文字是COBOL的保留字。在COBOL语言中，它们已有专门的含义，如ADD表示加法操作，MOVE表示传送操作，它们不能被用作它用。

大写的英文字下面如果划有横线，表示这个字是不能省略的，而是必写的。如果无横线，表示此英文字可写可不写。写的目的只是为了表达意思更清楚。如：

PICTURE IS 字符串

在具体使用时，写成，PICTURE 99 或 PICTURE IS 99 均可。IS是可以省略的。

(2) 有汉字的地方由程序设计者根据相应的语法规则填入具体内容。如：

ACCEPT 标识符

在“标识符”处应填入具体的内容，如：

ACCEPT A

A就是一个标识符(数据名)。

(3) 用方括号[　]括起来的部分是供程序设计者任选的，即它不是必要的成分，程序设计者根据实际需要决定对其取舍。如：

WRITE 记录名 [FROM 标识符]

具体用此语句时，可以写成：

WRITE ABC　　(将ABC记录输出)

或　WRITE ABC FROM T　　(先把数据名T的内容送到记录区ABC中，再将记录ABC输出)

即可以有两种形式的用法，适于不同的用途。

(4) 用花括号{　}括起来的部分，包括几个并列的内容，要求程序设计者根据需要从中选择其一。如：

$$\text{MOVE}\left\{\begin{matrix}\text{标识符 1}\\ \text{常量}\end{matrix}\right\}\text{TO 标识符 2}$$

可以在标识符和常量之中任选用一个，如：

MOVE A TO B (A，B是标识符)

或　MOVE 10 TO B (10是常量)

(5) 省略号…表示其左边第一个方括弧或花括弧中的内容可以重复多次。如：

$$\text{ADD}\left\{\begin{matrix}\text{标识符 1}\\ \text{常量 1}\end{matrix}\right\}\left[\begin{matrix}\text{，标识符 2}\\ \text{，常量 2}\end{matrix}\right]\cdots\text{TO 标识符 m}$$

在具体使用ADD语句时可以写成以下形式：

ADD A，B　　TO X

ADD A，B，C，D TO X

ADD A，B，C，D，E，F TO X

ADD 10，20，C，D TO X

即 ADD 后面可以跟两个以上的任意多个标识符或常数，A，B，C，D，E，F 均为标识符。

（6）格式中的字符“+”，“−”，“>”，“<”，“=”，虽然不带下线，但当用到它们时，必须写上而不能省略。

（7）格式中的“,”和“;”可以省略，如果写上它们，其后必须跟一空格。如上述加法语句中可以将“,”号省去，而写成：

ADD A B C D TO X

注意，A 和 B，B 和 C，C 和 D 之间都必须至少有一个空格。

# 附录Ⅲ　ANSI COBOL X 3.23　1974的语言格式表

标识部分

**标识部分一般格式**

IDENTIFICATION DIVISION.
PROGRAM-ID. 程序名.
[AUTHOR. [注解项] …]
[INSTALLATION. [注解项] …]
[DATE-WRITTEN. [注解项] …]
[DATE-COMPILED. [注解项] …]
[SECURITY. [注解项] …]

环境部分

**环境部分一般格式**

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 计算机名 [WITH DEBUGGING MODE].
OBJECT-COMPUTER. 计算机名.

[, MEMORY SIZE 整数 { WORDS / CHARACTERS / MODULES }]

[, PROGRAM COLLATING SEQUENCE IS 字母表名]

[, SEGMENT-LIMIT IS 段号]

[SPECIAL-NAMES.] 专用名

{ IS助忆名 [, ON STATUS IS 条件名 1 [, OFF STATUS IS 条件名 1]] /
IS助忆名 [, OFF STATUS IS 条件名 2 [, ON STATUS IS 条件名 2]] /
ON STATUS IS 条件名 1 [, OFF STATUS IS 条件名 2] /
OFF STATUS IS 条件名 2 [, ON STATUS IS 条件名 1] } …

```
{ 字母表名 IS { STANDARD-1
                NATIVE
                专用名
                常量 1 [ { THROUGH } 常量 2
                         { THRU    }
                         ALSO 常量 3 [, ALSO 常量 4] … ]
                常量 5 [ { THROUGH } 常量 6
                         { THRU    }
                         ALSO 常量 7 [, ALSO 常量 8] … ] … } }
    [, CURRENCY SIGN IS 常量 9]
    [, DECIMAL-POINT IS COMMA].
[INPUT OUTPUT SECTION.
FILE-CONTROL.
    {文件控制描述体} …
[I-O-CONTROL.
    [; RERUN [ON { 文件名 1 }]]
                 { 设备名  }
    EVERY { { [END OF] { REEL } } OF 文件名 2 } … ]
          {            { UNIT } }
          { 整数 1 RECORDS                    }
          { 整数 2 CLOCK-UNITS                }
          { 条件名                            }
    [; SAME { RECORD    } AREA FOR 文件名 3 {, 文件名 4} … ] …
            { SORT      }
            { SORT-MERGE }
    [; MULTIPLE FILE TAPE CONTAINS 文件名 5 [POSITION 整数 3]
    [, 文件名 6 [POSITION 整数 4]] …] …
```

**文件控制段一般格式**

**格式一**

```
SELECT [OPTIONAL] 文件名
    ASSIGN TO 文件名 1 [, 文件名 2] …
    [; RESERVE 整数 1 [ { AREA  } ]]
                        { AREAS }
    [; ORGANIZATION IS SEQUENTIAL]
    [; ACCESS MODE IS SEQUENTIAL]
    [; FILE STATUS IS 数据名 1].
```

**格式二**

```
SELECT 文件名
    ASSIGN TO 文件名 1 [, 文件名 2] …
```

( ; RESERVE 整数 1 ( AREA / AREAS ) )

; ORGANIZATION IS RELATIVE

( ; ACCESS MODE IS { SEQUENTIAL [, RELATIVE KEY IS 数据名 1] / ( RANDOM / DYNAMIC ), RELATIVE KEY IS 数据名 1 } )

[; FILE STATUS IS 数据名 2]

**格式三**

SELECT 文件名

ASSIGN TO 文件名 1 [, 文件名 2] …

( ; RESERVE 整数 1 ( AREA / AREAS ) )

; ORGANIZATION IS INDEXED

( ; ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC } )

; RECORD KEY IS 数据名 1

[; ALTERNATE RECORD KEY IS 数据名 2 [WITH DUPLICATES]] …

[; FILE STATUS IS 数据名 3].

**格式四**

SELECT 文件名 ASSIGN TO 文件名 1 [, 文件名 2] ….

## 数据部分

### 数据部分一般格式

DATA DIVISION.

FILE SECTION.

[FD 文件名

( ; BLOCK CONTAINS [整数 1 TO] 整数 2 { RECORDS / CHARACTERS } )

[; RECORD CONTAINS [整数 3 TO] 整数 4 CHARACTERS]

; LABEL { RECORD IS / RECORDS ARE } { STANDARD / OMITTED }

( ; VALUE OF 设备名 1 IS { 数据名 1 / 常量 1 } ( , 设备名 2 IS { 数据名 2 / 常量 2 } ) … )

( ; DATA { RECORD IS / RECORDS ARE } 数据名 3 [, 数据名 4] … )

( ; LINKAGE IS { 数据名 5 / 整数 5 } LINES ( , WITH FOOTING AT { 数据名 6 / 整数 6 } ) )

[, LINES AT TOP {数据名 7 / 整数 7}] [, LINES AT BOTTOM {数据名 8 / 整数 8}]]

[; CODE-SET IS 字母表名]

[; {REPORT IS / REPORTS ARE} 报表名 1[, 报表名 2] ….

[记录描述体] …] …

[SD 文件名 [; RECORD CONTAINS [整数 1 TO] 整数 2CHARACTERS]

[; DATA {RECORD IS / RECORDS ARE} 数据名 1 [, 数据名 2] …].

[记录描述体] …] …

[WORKING-STORAGE SECTION.

{77 层描述体 / 记录描述体} …]

[LINKAGE SECTION.

{77 层描述体 / 记录描述体} …]

[COMMUNICATION SECTION.

[通信描述体

[记录描述体] …] …]

[REPORT SECTION.

[RD 报表名

[; CODE 常量 1]

[; {CONTROL IS / CONTROLS ARE} {数据名 1 [, 数据名 2] … / , FINAL [, 数据名 1 [, 数据名 2] …]}]

[; PAGE {LIMIT IS / LIMITS ARE} 整数 1 {LINE / LINES} [, HEADING 整数 2]

[, FIRST DETAIL 整数 3] [, LAST DETAIL 整数 4]

[, FOOTING 整数 5]].

{报表性描述体} …] …]

## 数据描述体一般格式

### 格式一

层号 {数据名 1 / FILLER}

[; REDEFINES 数据名 2]

[; {PICTURE / PIC} IS 字符串]

[; [USAGE IS] {COMPUTATIONAL / COMP / DISPLAY / INDEX}]

[; [SIGN IS] {LEADING / TRAILING} [SEPARATE CHARACTER]]

[; OCCURS {整数 1 TO 整数 2 TIMES DEPENDING ON 数据名 3 / 整数 2 TIMES}

({ASCENDING / DESCENDING} KEY IS 数据名 4 [, 数据名 5] …) …

[INDEXED BY 位标名 1 [, 位标名 2] …]]

[; {SYNCHRONIZED / SYNC} {LEFT / RIGHT}]

[; {JUSTIFIED / JUST} RIGHT]

[; BLANK WHEN ZERO]

[; VALUE IS 常量].

**格式二**

66 数据名 1; RENAMES 数据名 2 [{THROUGH / THRU} 数据名 3]

**格式三**

88 条件名; {VALUE IS / VALUES ARE} 常量 1 [{THROUGH / THRU} 常量 2]

[, 常量 3 [{THROUGH / THRU} 常量 4]] …

**通信描述体一般格式**

**格式一**

通信描述名 FOR [INITIAL] INPUT

[[ [; SYMBOLIC QUEUE IS 数据名 1]
[; SYMBOLIC SUB-QUEUE-1 IS 数据名 2]
[; SYMBOLIC SUB-QUEUE-2 IS 数据名 3]
[; SYMBOLIC SUB QUEUE-3 IS 数据名 4]
[; MESSAGE DATE IS 数据名 5]
[; MESSAGE TIME IS 数据名 6]
[; SYMBOLIC SOURCE IS 数据名 7]
[; TEXT LENGTH IS 数据名 8]
[; END KEY IS 数据名 9]
[; STATUS KEY IS 数据名 10]
[; MESSAGE COUNT IS 数据名 11]]
[数据名 1, 数据名 2, …, 数据名 11]]

**格式二**

通信描述名; FOR OUTPUT

[; DESTINATION COUNT IS 数据名 1]
[; TEXT LENGTH IS 数据名 2]
[; STATUS KEY IS 数据名 3]
[; DESTINATION TABLE OCCURS 整数 2 TIMES
[; INDEXED BY 位标名 1 [, 位标名 2] …]]
[; ERROR KEY IS 数据名 4]
[; SYMBOLIC DESTINATION IS 数据名 5].

**报表栏描述体一般格式**

**格式一**

01 [数据名 1]

[; LINE NUMBER IS { 整数 1 [ON NEXT PAGE] / PLUS 整数 2 }]

[; NEXT GROUP IS { 整数 3 / PLUS 整数 4 / NEXT PAGE }]

, TYPE IS [ { REPORT HEADING / RH } / { PAGE HEADING / PH } / { CONTROL HEADING / CH } { 数据名 2 / FINAL } / { DETAIL / DE } ]

[ { CONTROL FOOTING / CF } { 数据名 3 / FINAL } / { PAGE FOOTING / PF } / { REPORT FOOTING / RF } ]

[; [USAGE IS] DISPLAY].

**格式二**

层号 [数据名 1]

[; LINE NUMBER IS { 整数 1 [ON NEXT PAGE] / PLUS 整数 2 }]

[; [USAGE IS] DISPLAY].

**格式三**

层号 [数据名 1]

[; BLANK WHEN ZERO]

[；GROUP INDICATE]

[；{JUSTIFIED / JUST} RIGHT]

[[；LINE NUMBER IS {整数 1 [ON NEXT PAGE] / PLUS 整数 2}]]

[；COLUMN NUMBER IS 整数 3]

；{PICTURE / PIC} IS 字符串

{
；SOURCE IS 标识符 1
；VALUE IS 常量
{；SUM 标识符 2 [，标识符 3] …
[UPON 数据名 2 [，数据名 3] …]} …
[RESET ON {数据名 4 / FINAL}]
}

[；[USAGE IS] DISPLAY].

过程部分

## 过程部分一般格式

### 格式一

PROCEDURE DIVISION [USING 数据名 1 [，数据名 2] …].
[DECLARATIVES.
{节名 SECTION [段号]. USE 句子
[段句. [句子] …] …} …

### 格式二

PROCEDURE DIVISION [USING 数据名 1 [，数据名 2] …].
{段名. [句子] …} …

## 过程部语句的一般格式

ACCEPT 标识符 [FROM助忆名]

ACCEPT 标识符 FROM {DATE / DAY / TIME}

ACCEPT 数据描述符名 MESSAGE COUNT

ADD {标识符 1 / 常量 1} [，标识符 2 / ，常量 2] …TO 标识符 m [ROUNDED]
[，标识符 n [ROUNDED]] … [；ON SIZE ERROR 强制语句]

ADD {标识符 1 / 常量 1}，{标识符 2 / 常量 2} [，标识符 3 / ，常量 3] …
GIVING 标识符 m [ROUNDED] [，标识符 n [ROUNDED]] …

[；ON SIZE ERROR] 强制语句]

ADD {CORRESPONDING / CORR} 标识符 1 TO标识符 2 [ROUNDED]

[；ON SIZE ERROR 强制语句]

ALTER 过程名 1 TO [PROCEED TO] 过程名 2

[；过程名 3 TO [PROCEED TO] 过程名 4] …

CALL {标识符 1 / 常量 1} [USING 数据名 1 [，数据名 2] …]

[ON OVERFLOW强制语句]

CANCEL {标识符 1 / 常量 1} [，标识符 2 / ，常量 2]…

CLOSE文件名 1 [ {REEL / UNIT} [WITH NO REWIND / FOR REMOVAL] / WITH {NO REWIND / LOCK} ]

[，文件 2 [ {REEL / UNIT} [WITH NO REWIND / FOR REMOVAL] / WITH {NO REWIND / LOCK} ]]

CLOSE 文件名 1 [WITH LOCK] [，文件名 2 [WITH LOCK]] …

COMPUTE标识符 1 [ROUNDED] [，标识符 2 [ROUNDED]] …

=算术表达式 [；ON SIZE ERROR 强制语句]

DELETE文件名 RECORD [；INVALID KEY 强制语句]

DISABLE {INPUT [TERMINAL] / OUTPUT} 通信描述名 WITH KEY {标识符 1 / 常量 1}

DISPLAY {标识符 1 / 常量 1} [标识符 2 / 常量 2]…[UPON 助忆名]

DIVIDE {标识符 1 / 常量 1} INTO 标识符 2 [ROUNDED]

[，标识符 3 [ROUNDED]] … [；ON SIZE ERROR 强制语句]

DIVIDE {标识符 1 / 常量 1} INTO {标识符 2 / 常量 2} GIVING 标识符 3 [ROUNDED]

[，标识符 4 [ROUNDED]] … [；ON SIZE ERROR 强制语句]

DIVIDE {标识符1 / 常量1} BY {标识符2 / 常量2} GIVING标识符3 [ROUNDED]

[，标识符4 [ROUNDED]] … [；ON SIZE ERROR 强制语句]

DIVIDE {标识符1 / 常量1} INTO {标识符2 / 常量2} GIVING 标识符3 [ROUNDED]

REMAINDER 标识符4 [；ON SIZE ERROR 强制语句]

DIVIDE {标识符1 / 常量1} BY {标识符2 / 常量2} GIVING标识符3 [ROUNDED]

REMAINDER 标识符4 [；ON SIZE ERROR 强制语句]

ENABLE {INPUT [TERMINAL] / OUTPUT} 通信描述名 WITH KEY {标识符1 / 常量1}

ENTER语言名 [例行程序名]

EXIT [PROGRAM]

GENERATE { 数据名 / 报表名 }

GO TO [过程名]

GO TO 过程名1 [，过程名2] …，过程名 n DEPENDING ON 标识符

IF条件；{ 语句1 / NEXT SENTENCE } { ；ELSE 语句2 / ；ELSE NEXT SENTENCE }

INITIATE报表名1 [，报表名2] …

INSPECT标识符1TALLYING

{ ，标识符2 FOR { ，{ { ALL / LEADING } { 标识符3 / 常量1 } / CHARACTERS } }

[ { BEFORE / AFTER } INITIAL { 标识符4 / 常量2 } ] }… }…

INSPECT标识符1 REPLACING

{ CHARACTERS BY { 标识符6 / 常量4 } [ { BEFORE / AFTER } INITIAL { 标识符7 / 常量5 } ]

{ { ，{ ALL / LEADING / FIRST } } { ，{ 标识符5 / 常量3 } BY { 标识符6 / 常量4 }

[ { BEFORE / AFTER } INITIAL { 标识符7 / 常量5 } ] }… }…

INSPECT标识符1 TALLYING

{ ，标识符2 FOR { ，{ { ALL / LEADING } { 标识符3 / 常量1 } / CHARACTERS } }

[ { BEFORE / AFTER } INITIAL { 标识符4 / 常量2 } ] }… }…

REPLACING

{ CHARACTERS BY { 标识符6 / 常量4 } [ { BEFORE / AFTER } INITIAL { 标识符7 / 常量5 } ]

{ { ，{ ALL / LEADING / FIRST } } { ，{ 标识符5 / 常量3 } BY { 标识符6 / 常量4 }

[ { BEFORE / AFTER } INITIAL { 标识符7 / 常量5 } ] }… }… }

MERGE文件名1 ON { ASCENDING / DESCENDING } KEY 数据名1 [，数据名2] …

[ ON { ASCENDING / DESCENDING } KEY 数据名3 [，数据名4] … ]…

[COLLATING SEQUENCE IS 字母表名]

USING文件名2，文件名3 [，文件名4] …

{OUTPUT PROCEDURE IS 节名1 [{THROUGH / THRU} 节名2]
 GIVING 文件名5}

MOVE {标识符1 / 常量} TO 标识符2 [，标识符3] …

MOVE {CORRESPONDING / CORR} 标识符1 TO 标识符2

MULTIPLY {标识符1 / 常量1} BY 标识符2 [ROUNDED]
[，标识符3 [ROUNDED]] … [；ON SIZE ERROR 强制语句]

MULTIPLY {标识符1 / 常量1} BY {标识符2 / 常量2} GIVING 标识符3 [ROUNDED]
[，标识符4 [ROUNDED]] … [；ON SIZE ERROR 强制语句]

OPEN {
 INPUT 文件名1 [{REVERSED / WITH NO REWIND}]
  [，文件名2 [{REVERSED / WITH NO REWIND}]] …
 OUTPUT 文件名3 [WITH NO REWIND] [，文件名4 [WITH NO REWIND]] …
 I-O 文件名5 [，文件名6] …
 EXTEND 文件名7 [，文件名8] …
} …

OPEN {
 INPUT 文件名1 [，文件名2] …
 OUTPUT 文件名3 [，文件名4] …
 I-O 文件名5 [，文件名6] …
} …

PERFORM 过程名1 [{THROUGH / THRU} 过程名2]

PERFORM 过程名1 [{THROUGH / THRU} 过程名2] {标识符1 / 常量1} TIMES

PERFORM 过程名1 [{THROUGH / THRU} 过程名2] UNTIL 条件1

PERFORM 过程名1 [{THROUGH / THRU} 过程名2]

VARYING {标识符2 / 位标名1} FROM {标识符3 / 位标名2 / 常量1} BY {标识符4 / 常量3} UNTIL 条件1
[AFTER {标识符5 / 位标名3} FROM {标识符6 / 位标名4 / 常量3} BY {标识符7 / 常量4} UNTIL 条件2]
[AFTER {标识符8 / 位标名5} FROM {标识符9 / 位标名6 / 常量5} BY {标识符10 / 常量6} UNTIL 条件3]

READ 文件名 RECORD [INTO 标识符] [；AT END 强制语句]

READ 文件名 [NEXT] RECORD [INTO 标识符] [；AT END 强制语句]

READ 文件名 RECORD [INTO 标识符] [；INVALID KEY 强制语句]

READ文件名 RECORD [INTO 标识符] [；KEY IS 数据名]
[，INVALID KEY 强制语句]

RECEIVE 通信描述名 {MESSAGE / SEGMENT} INTO标识符1 [；N O DATA 强制语句]

RELEASE 记录名 [FROM 标识符]

RETURN 文件名 RECORD [INTO 标识符] [；AT END 强制语句]

REWRITE 记录名 [FROM 标识符]

REWRITE 记录名 [FROM标识符] [；INVALID KEY 强制语句]

SEARCH 标识符1 [VARYING {标识符2 / 位标名1}] [；AT END 强制语句1]
WHEN条件1 {强制语句2 / NEXT SENTENCE}
[；WHEN条件2 {强制语句3 / NEXT SENTENCE}]…

SEARCH ALL 标识符1 [；AT END 强制语句1]
；WHEN {数据名1 {IS EQUAL TO / IS= } {标识符3 / 常量1 / 算术表达式1} / 条件名1}
[AND {数据名2 {IS EQUAL TO / IS= } {标识符4 / 常量2 / 算术表达式2} / 条件名2}]…
{强制语句2 / NEXT SENTENCE}

SEND通信描述名FROM标识符1

SEND通信描述名 [FROM 标识符1] {WITH 标识符2 / WITH ESI / WITH EMI / WITH EGI}
[{BEFORE / AFTER} ADVANCING {{{标识符3 / 整数1} [LINE / LINES]} / {助忆名 / PAGE}}]

SET {标识符1 [，标识符2] … / 位标1 [，位标2] …} TO {标识符3 / 位标3 / 整数1}

SET位标4 [，位标5] … {UP BY / DOWN BY} {标识符4 / 常量2}

SORT文件名1 ON {ASCENDING / DESCENDING} KEY 数据名1 [，数据名2] …
[ON {ASCENDING / DESCENDING} KEY 数据名3 [，数据名4] …]…
[COLLATING SEQUENCE IS 字母表名]
{INPUT PROCEDURE IS 节名1 [{THROUGH / THRU} 节名2] / USING 文件名2 [，文件名3] …}

{OUTPUT PROCEDURE IS 节名3 [{THROUGH / THRU} 节名4]
 GIVING 文件名4}

START 文件名 [KEY {IS EQUAL TO / IS= / IS GREATER THAN / IS> / IS NOT LESS THAN / IS NOT<} 数据名]
[; INVALID KEY 强制语句]

STOP {RUN / 常量}

STRING {标识符1 / 常量1} [, {标识符2 / 常量2}]…DELIMITED BY {标识符3 / 常量3 / SIZE}
[, {标识符4 / 常量4} [, {标识符5 / 常量5}]…DELIMITED BY {标识符6 / 常量6 / SIZE}]
 INTO 标识符7 [WITH POINTER 标识符8] [; ON OVERFLOW 强制语句]

SUBTRACT {标识符1 / 常量1} [, {标识符2 / 常量2}]…FROM 标识符 m [ROUNDED]
 [标识符 n [ROUNDED]] … [; ON SIZE ERROR 强制语句]

SUBTRACT {标识符1 / 常量1} [, {标识符2 / 常量2}]…FROM {标识符 m / 常量 m}
 GIVNG 标识符 n [ROUNDED] [, 标识符 o [ROUNDED]] …
 [; ON SIZE ERROR 强制语句]

SUBTRACT {CORRESPONDING / CORR} 标识符1 FROM 标识符2 [ROUNDED]
 [; ON SIZE ERROR 强制语句]

SUPPRESS PRINTING

TERMINATE 报表名1 [报表名2] …

UNSTRING 标识符1
 DELIMITED BY [ALL] {标识符2 / 常量1} [, OR [ALL] {标识符3 / 常量2}]…]
 INTO 标识符4 [, DELIMITER IN 标识符5] [, COUNT IN 标识符6]
 [, 标识符7 [DELIMITER IN 标识符8] [, COUNT IN 标识符9] …]
 [WITH POINTER 标识符10] [TALLYING IN 标识符11]
 [; ON OVERFLOW 强制语句]

USE AFTER STANDARD {EXCEPTION / ERROR} PROCEDURE ON
 {文件名1 [文件名2] … / INPUT / OUTPUT / I—O / EXTEND}

USE AFTER STANDARD {EXCEPTION / ERROR} PROCEDURE ON

{文件名1 [，文件名2] … / INPUT / OUTPUT / I—O}

USE BEFORE REPORTING 标识符

USE FOR DEBEGINNING ON {通信描述名1 / [ALL REFERENCES OF] 标识符1 / 文件名1 / 过程名1 / ALL PROCEDURES}

[通信描述名2 / [ALL REFERENCES OF] 标识符2 / 文件名2 / 过程名2 / ALL PROCEDURES]

WRITE 记录名 [FROM 标识符1]

[{BEFORE / AFTER} ADVANCING {{标识符2 / 整数1} [LINE / LINES] / {助忆名 / PAGE}}]

[；AT {END-OF-PAGE / EOP} 强制语句]

WRITE记录名 [FROM 标识符] [；INVALID KEY 强制语句]

### 条件的一般格式

### 关系表达式条件

{标识符1 / 常量1 / 算术表达式1 / 位标名1} {IS [NOT] GREATER THAN / IS [NOT] LESS THAN / IS [NOT] EQUAL TO / IS [NOT] > / IS [NOT] < / IS [NOT] =} {标识符2 / 常量2 / 算术表达式2 / 位标名2}

### 类型条件

标识符[NOT] {NUMERIC / ALPHABETIC}

### 符号条件

算术表达式 IS [NOT] {POSITIVE / NEGATIVE / ZERO}

**条件名条件：**

条件名

**开关状态条件：**

条件名

**否定(非)简单条件：**

NOT 简单条件

**组合条件：**

条件 { { AND / OR } 条件 } …

**压缩组合关系表达式条件：**

关系表达式条件 { { AND / OR } [NOT] [关系运算符] 客体 } …

**其它格式**

**限定**

{ 数据名1 / 条件名 } [ { OF / IN } 数据名2 ] …

段名 [ { OF / IN } 节名 ]

[ { OF / IN } 库名 ]

**下标**

{ 数据名 / 条件名 } (下标1 [，下标2 [，下标3]])

**位标**

{ 数据名 / 条件名 } { { 位标名1 [ {±} 常量2] / 常量1 } [，{ 位标名2 [ {±} 常量4] / 常量3 } [，{ 位标名3 [ {±} 常量6] / 常量5 }]] )

**标识符**

**格式一**

数据名1 [ { OF / IN } 数据名2 ] … [ (下标1 [，下标2 [，下标3]])]

**格式二**

数据名1[{OF/IN}数据名2]…[({位标名1 [(±) 常量2]/常量1}
[,{位标名2 [(±) 常量4]/常量3}[,{位标名3 [(±) 常量6]/常量5}]])]

**COPY 语句的一般格式**

COPY正文名[{OF/IN}库名]
[REPLACING{,{伪正文1/标识符1/常量1/字1} BY {伪正文2/标识符2/常量2/字2}}…]

# 附录Ⅳ　简单的 COBOL 程序的格式索引

为了使初学者在编写简单程序中查阅方便，我们列出最常用的、最简单的语法格式供参考。如果用到本索引未包括的内容，可查附录Ⅲ的详细格式。

IDENTIFICATION DIVISION.（标识部）
PROGRAM-ID. 程序名.
ENVIRONMENT DIVISION.（环境部）
INPUT-OUTPUT SECTION.（输入输出节）
FILE-CONTROL.　　（文件控制段）
　SELECT 文件名 ASSIGN TO 设备名.
DATA DIVISION.（数据部）
FILE SECTION.（文件节）
FD文件名
　[BLOCK CONTAINS 整数RECORDS]
　LABEL RECORD IS {OMITTED/STANDARD}
　[DATA RECORD IS 数据名].
01 数据名.
层号{数据名/FILLER}[REDEFINES 数据名]
　[OCCURS整数 TIMES][{PICTURE/PIC} IS 字符串].
WORKING-STORAGE SECTION.（工作单元节）
77 数据名{PICTURE/PIC} IS 字符串 [VALUE IS 常量 ].
01 数据名.
层号{数据名/FILLER}[REDEFINES 数据名][OCCURS 整数 TIMES]

[{PICTURE / PIC} IS 字符串] [VALUE IS 常量]

PROCEDURE DIVISION. (过程部)

段名.

OPEN {INPUT 文件名1 [, 文件名2] … / OUTPUT 文件名3 [, 文件名4] …} …

READ 文件名 [; AT END 强制语句]

IF条件 强制语句1 [; ELSE 强制语句2]

MOVE [CORR] {标识符1 / 常量} TO 标识符2

COMPUTE 标识符 = 算术表达式

WRITE 记录名

[{BEFORE / AFTER} ADVANCING {标识符 / 整数 / PAGE} {LINE / LINES}]

PERFORM 过程名1 [THRU 过程名2]

GO TO 过程名

SORT 文件名1 ON {ASCENDING / DESCENDING} KEY 数据名1 [, 数据名2] …

[ON {ASCENDING / DESCENING} KEY 数据名3 [, 数据名4] …] …

{INPUT PROCEDURE IS 节名1 [THRU 节名2] / USING 文件名2}

{OUTPUT PROCEDURE IS 节名3 [THRU 节名4] / GIVING 文件名3}

CLOSE 文件名1 [文件名2] …

STOP RUN.

# 附录 V　COBOL 保留字表

| | | |
|---|---|---|
| ACCEPT | ASSIGN | CH |
| ACCESS | AT | CHARACTER |
| ADD | AUTHOR | CHARACTERS |
| ADVANCING | | CLOCK-UNITS |
| AFTER | BEFORE | CLOSE |
| ALL | BLANK | COBOL |
| ALPHABETIC | BLOCK | CODE |
| ALSO | BOTTOM | CODE-SET |
| ALTER | BY | COLLATING |
| ALTERNATE | | COLUMN |
| AND | CALL | COMMA |
| ARE | CANCEL | COMMUNICATION |
| AREA | CD | COMP |
| AREAS | CF | COMPUTATIONAL |
| ASCENDING | | COMPUTE |

CONFIGURATION
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

ELSE
EMI
ENABLE
END
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
ESI
EVERY
EXCEPTION
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
I-O
I-O-CONTROL
IS

JUST
JUSTIFIED

KEY

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINES
LINE-COUNTER
LINKAGE
LOCK
LOW-VALUE
LOW VALUES

MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLY
MULTIPLE

NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH

PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PROCEDURE
PROCEED
PROGRAM
PROGRAM-ID

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
RECEIVE
RECORD
RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMAINDER
REMOTE
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
START
STATUS
STOP
STRING
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONIZED

TABLE
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TYPE

UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

说明：本表基本上列出了 ANSI COBOL 1974 的保留字。各种计算机系统所用的 COBOL 保留字可能与本表中列出的有一些差别。例如，ADDRESS 在 COBOL 1974 中不是保留字，而在某些计算机 COBOL 中作为保留字。使用时应查所用计算机系统 COBOL 说明书。本表只供参考。

# 附录Ⅵ　ANSI COBOL X 3.23-1985的语言格式表

## 标识部

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program name [ IS { COMMON | INITIAL } PROGRAM ].
[AUTHOR. [comment-entry] ...]
[INSTALLATION. [comment-entry] ...]
[DATE-WRITTEN. [comment-entry] ...]
[DATE-COMPILED. [comment-entry] ...]
[SECURITY. [comment-entry] ...]
```

## 环境部

```
ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
 [SOURCE-COMPUTER. [source-computer-entry]]
 [OBJECT-COMPUTER. [object-computer-entry]]
 [SPECIAL-NAMES. [special-names-entry]]]
[INPUT-OUTPUT SECTION.
 FILE-CONTROL. {file-control-entry} ...
 [I-O-CONTROL. [input-output-control-entry]]]
```

## 源计算机段

```
SOURCE-COMPUTER.
[computer-type [WITH DEBUGGING MODE]]
```

## 目标计算机段

```
OBJECT-COMPUTER.
[computer-type
 [MEMORY SIZE integer { WORDS | CHARACTERS | MODULES }]
 [PROGRAM COLLATING SEQUENCE IS alpha-name]
 [SEGMENT-LIMIT IS segment-number]]
```

## 专用名描述栏目

```
SPECIAL-NAMES.
```

```
[ { CARD-READER
  | PAPER-TAPE-READER
  | CONSOLE
  | LINE-PRINTER
  | PAPER-TAPE-PUNCH
  | C01 } IS dlvice-name
SWITCH switch-num
  { IS switch-name
  | [ON STATUS IS cond-name]
  | [OFF STATUS IS cond name]
  | IS switch-name
  | [OFF STATUS IS cond-name]
  | [ON STATUS IS cond-name]
  | ON STATUS IS cond-name [OFF STATUS IS cond-name]
  | OFF STATUS IS cond name [ON STATUS IS cond name] } ]
[ALPHABET alpha-name IS
  { STANDARD-1
  | STANDARD-2
  | NATIVE
  | { first-literal [ { THRU    } last-literal ]
    |                { THROUGH }                 } ... } ... ]
[SYMBOLIC CHARACTERS
  { { {symbol-char} ... { IS  } {char-val} ... } ... [IN alpha-name] } ... } ... ]
                        { ARE }
[CLASS class-name IS
  { first-literal [ { THRU    } last-literal ] } ... } ... ]
                  { THROUGH }
[CURRENCY SIGN IS char]
[DECIMAL-POINT IS COMMA]. ]
```

### 文件控制段

```
FILE-CONTROL.
```

### 格式一——顺序文件

```
SELECT [OPTIONAL] file-name
  ASSIGN TO file-spec
[RESERVE reserve-num [ AREA ]]
                     [ AREAS ]
[[ORGANIZATION IS] SEQUENTIAL]
[PADDING CHARACTER IS pad-char]
[RECORD DELIMITER IS STANDARD-1]
```

[ACCESS MODE IS SEQUENTIAL]

[FILE STATUS IS file-stat]

### 格式二 —— 相对文件

```
SELECT [OPTIONAL] file-name
    ASSIGN TO file-spec
[RESERVE reserve-num [AREA | AREAS]]
[ORGANIZATION IS] RELATIVE
[RECORD DELIMITER IS STANDARD-1]
[ACCESS MODE IS {SEQUENTIAL [RELATIVE KEY IS rel-key] | {RANDOM | DYNAMIC} RELATIVE KEY IS rel-key}]
[FILE STATUS IS file-stat]
```

### 格式三 —— 索引文件

```
SELECT [OPTIONAL] file-name
    ASSIGN TO file-spec
[RESERVE reserve-num [AREA | AREAS]]
[ORGANIZATION IS] INDEXED
[RECORD DELIMITER IS STANDARD-1]
[ACCESS MODE IS {SEQUENTIAL | RANDOM | DYNAMIC}]
[RECORD KEY IS rec-key]
[ALTERNATE RECORD KEY IS alt-key [WITH DUPLICATES] ...
[FILE STATUS IS file-stat]
```

### 格式四 —— 排序/合并文件

```
SELECT file-name ASSIGN TO file-spec.
```

### 格式五 —— 报表文件

```
SELECT file-name
    ASSIGN TO file-spec
[RESERVE reserve-num [AREA | AREAS]]
[[ORGANIZATION IS] SEQUENTIAL]
[PADDING CHARACTER IS pad-char]
```

[RECORD DELIMITER IS STANDARD 1]
[ACCESS MODE IS SEQUENTIAL]
[FILE STATUS IS file-stat]

### 输入-输出控制段

```
I-O-CONTROL. [
[SAME
 [RECORD
  SORT
  SORT-MERGE] AREA FOR {same-area-file} {same-area-file} ...] ...
[RERUN
 [ON file-name] EVERY
 {{{[END OF] {REEL}
  {integer RECORDS}} OF file-name
  integer CLOCK-UNITS
  condition-name}]
[MULTIPLE FILE TAPE CONTAINS
    {file-name [POSITION integer]} ...] ... .]
```

### 数据部

```
DATA DIVISION.
[FILE SECTION.
 [file-description-entry {record-description-entry} ...] ...
 [report file-description-entry] ...
 [sort merge-file-description-entry {record-description-entry} ...] ...]
[WORKING STORAGE SECTION. [record-description-entry] ...]
[LINKAGE SECTION. [record-description-entry] ...]
[REPORT SECTION. [report-description-entry
                  {report group-description-entry} ...]]
```

### 文件描述栏目

### 格式一 ——顺序文件

```
FD file-name
[IS EXTERNAL]
[IS GLOBAL]
[BLOCK CONTAINS [smallest-block TO] blocksize {RECORDS
                                               CHARACTERS}]
```

```
[ RECORD { CONTAINS [ shortest-rec TO ] longest - rec CHARACTERS }
         { IS VARYING IN SIZE                                    }
         { [FROM shortest - rec] [TO longest - rec] CHARACTERS   }
         { [DEPENDING ON depending - item]                       } ]

[ LABEL { RECORDS ARE } { STANDARD } ]
        { RECORD IS   } { OMITTED  }

[VALUE OF ID IS file - spec]

[ DATA { RECORDS ARE } {rec - name } ... ]
       { RECORD IS   }

[ LINAGE IS page - size LINES                    ]
    [WITH FOOTING AT footing line]
    [LINES AT TOP top lines]
    [LINES AT BOTTOM bottom - lines]

[ CODE - SET IS alpha name ]
```

**格式二-- 相对文件**

```
FD file - name

[ IS EXTERNAL]

[ IS GLOBAL]

[ BLOCK CONTAINS [ smallest - block TO ] blocksize { RECORDS    } ]
                                                   { CHARACTERS }

[ RECORD ] { CONTAINS [ shortest rec TO ] longest - rec CHARACTERS }
           { IS VARYING IN SIZE                                    }
           { [FROM shortest - rec] [ TO longest rec ] CHARACTERS   }
           { [ DEPENDING ON depending item]                        }

[ LABEL { RECORDS ARE } { STANDARD } ]
        { RECORD IS   } { OMITTED  }

[ VALUE OF ID IS file spec ]

[ DATA { RECORDS ARE } { rec - name } ... ]
       { RECORD IS   }
```

**格式三——索引文件**

```
FD file - name

[ IS EXTERNAL]
[ IS GLOBAL]

[ BLOCK CONTAINS [ smallest - block TO ] blocksize { RECORDS    } ]
                                                   { CHARACTERS }
```

```
[ RECORD { CONTAINS [ shortest - rec TO ] longest - rec CHARACTERS
         { IS VARYING IN SIZE
           [ FROM shortest - rec ] [ TO longest - rec ] CHARACTERS
           [ DEPENDING ON depending - item ] } ]

[ LABEL { RECORDS ARE } { STANDARD } ]
        { RECORD IS   } { OMITTED  }

[ VALUE OF ID IS file - spec ]

[ DATA { RECORDS ARE } { rec - name } ... ]
       { RECORD IS   }
```

**格式四－　报表文件**

```
FD  file - name

[ IS EXTERNAL ]

[ IS GLOBAL ]

[ BLOCK CONTAINS [ smallest - block TO ] blocksize { RECORDS    } ]
                                                   { CHARACTERS }

[ RECORD { CONTAINS [ shortest - rec TO ] longest - rec ] CHARACTERS
         { IS VARYING IN SIZE
           [ FROM shortest - rec ] [ TO longest - rec ] CHARACTERS
           [ DEPENDING ON depending - item ] } ]

[ LABEL { RECORDS ARE } { STANDARD } ]
        { RECORD IS   } { OMITTED  }

[ VALUE OF ID IS file - spec ]

{ REPORT IS  } { report - name } ...
{ REPORTS ARE }

[ CODE - SET IS alpha - name ].
```

**排序/合并文件描述栏目**

```
SD  file - name

[ RECORD { CONTAINS [ shortest - rec TO ] longest - rec CHARACTERS
         { IS VARYING IN SIZE
           [ FROM shortest - rec ] [ TO longest - rec ] CHARACTERS
           [ DEPENDING ON depending - item ] } ]

[ DATA { RECORDS ARE } { rec - name } ... ].
       { RECORD IS   }
```

**报表文件描述栏目**

```
RD  report - name
```

```
[IS GLOBAL]
[CODE report code]
[{CONTROL IS / CONTROLS ARE} {{control name} ... / FINAL [control - name] ...}]
[PAGE [{LIMIT IS / LIMITS ARE}] page - size [{LINE / LINES}]
    [HEADING heading - line]
    [FIRST DETAIL first detail - line]
    [LAST DETAIL last - detail - line]
    [FOOTING footing - line]]
```

## 数据描述栏目

### 格式一

```
level - number [{data - name / FILLER}]
[REDEFINES other - data - item]
[IS EXTERNAL]
[IS GLOBAL]
[{PICTURE / PIC} IS character - string]
[[USAGE IS] {BINARY / {COMPUTATIONAL / COMP} / DISPLAY / INDEX / PACKED - DECIMAL}]
[[SIGN IS] {LEADING / TRAILING} [SEPARATE CHARACTER]]
[OCCURS table - size TIMES
    [{ASCENDING / DESCENDING} KEY IS {key - name} ...]...
    [INDEXED BY {ind - name} ...]
 OCCURS min - times TO max times TIMES DEPENDING ON
                                        depending - item
    [{ASCENDING / DESCENDING} KEY IS {key - name} ...]...
    [INDEXED BY {ind name} ...]]
[{SYNCHRONIZED / SYNC} [{LEFT / RIGHT}]]
[{JUSTIFIED / JUST} RIGHT]
```

[ BLANK WHEN ZERO]
[ VALUE IS literal].

### 格式二

66 new — name RENAMES rename -- start [ { THRU / THROUGH } rename -- end ].

### 格式三

88 condition — name
{ VALUE IS / VALUES ARE }
{ low -- value [ { THRU / THROUGH } high -- value ] }.

### 报表文件组描述栏目

(Report Group Descriptions)

### 格式一

01 [ group — data — name]

[ LINE NUMBER IS { line — num [ ON NEXT PAGE ] / PLUS line — num — plus } ]

[ NEXT GROUP IS { next — group — line — num / PLUS next — group — line — num — plus / NEXT PAGE } ]

TYPE IS { { REPORT HEADING } / RH
{ PAGE HEADING } / PH
{ CONTROL HEADING } / CH { control — head — name } / FINAL
{ DETAIL } / DE
{ CONTROL FOOTING } / CF { control — foot — name } / FINAL
{ PAGE FOOTING } / PF
{ REPORT FOOTING } / RF }

[ [USAGE IS] DISPLAY ].

### 格式二

```
level - number [ group — data - name]

[LINE NUMBER IS {line — num [ON NEXT PAGE]}]
               {PLUS line — num - plus    }

[[ USAGE IS] DISPLAY].
```

### 格式三

```
level number [ group - data — name ]

[BLANK WHEN ZERO]
[COLUMN NUMBER IS column num]
[GROUP INDICATE ]

[{JUSTIFIED} RIGHT]
 {JUST     }

[LINE NUMBER IS {line — num [ON NEXT PAGE]}]
               {PLUS line — num — plus    }

{PICTURE} IS character — string
{PIC    }

[[ SIGN IS] {LEADING } SEPARATE CHARACTER]
            {TRAILING}

{SOURCE IS source — name                                                                    }
{VALUE IS lit                                                                               }
{{ SUM (sum — name) ... [ UPON ( detail - report group name) ...]}...                       }
{    [RESET ON {control - foot — name}]                                                     }
{              {FINAL                }                                                    }

[ [ USAGE IS] DISPLAY].
```

### 过程部

### 格式一

```
PROCEDURE DIVISION [ USING (data — name) ...].

[DECLARATIVES.
 {section — name SECTION [segment — number].          }...
 {  declarative sentence [ paragraph — name.          }
 {   [ sentence ] ...] ...                            }
END DECLARATIVES.]
```

{section -- name SECTION [ segment — number].
[ paragraph — name . [sentence] …] …} …

**格式二**

PROCEDURE DIVISION [ USING {data - name} …].
[paragraph — name. [ sentence ] …] …

**ACCEPT 语句**

**格式一**

ACCEPT dest item [FROM input - source]

**格式二**

ACCEPT dest item FROM {DATE / DAY / DAY—OF—WEEK / TIME}

**ADD 语句**

**格式一**

ADD {num } … TO { rsult [ ROUNDED]} …
[ ON SIZE ERROR stment]
[ NOT ON SIZE ERROR stment2]
[ END—ADD]

**格式二**

ADD {num} … TO {num} GIVING { rsult [ ROUNDED ]} …
[ ON SIZE ERROR stment]
[ NOT ON SIZE ERROR stment 2]
[END -ADD]

**格式三**

ADD {CORRESPONDING / CORR} grp - 1 TO grp—2 [ROUNDED]
[ ON SIZE ERROR stment]
[ NOT ON SIZE ERROR stment2]
[ END—ADD]

**ALTER 语句**

ALTER { proc TO [ PROCEED TO] new — proc } …

## CALL 语句

```
CALL prog - name
[USING
    {{[BY REFERENCE]
     BY CONTENT
     BY DESCRIPTOR
     BY VALUE} {arg} ...} {{BY REFERENCE
                            BY CONTENT} {arg} ...}]
{[ON EXCEPTION stment] [NOT ON EXCEPTION stment2]
 [ON OVERFLOW stment]}
[END-CALL]
```

## CANCEL 语句

```
CANCEL {prog - name} ...
```

## CLOSE 语句

```
CLOSE file-name [{REEL} [FOR REMOVAL]
                 {UNIT} [WITH NO REWIND]
                WITH {NO REWIND} ...
                     {LOCK}
```

## COMPUTE 语句

```
COMPUTE {result [ROUNDED]} ... = arithmetic - expression
    [ON SIZE ERROR stment]
    [NOT ON SIZE ERROR stment2]
    [END-COMPUTE]
```

## CONTINUE 语句

```
CONTINUE
```

## DELETE 语句

```
DELETE file name RECORD
    [INVALID KEY stment]
    [NOT INVALID KEY stment2]
    [END DELETE]
```

## DISPLAY 语句

### 格式

```
DISPLAY [src - item] ...
```

[ UPON output—dest ] [ WITH NO ADVANCING]

**DIVIDE 语句**

**格式一**

DIVIDE srcnum INTO {rsult [ROUNDED]} …

[ON SIZE ERROR stment]

[NOT ON SIZE ERROR stment2]

[END DIVIDE]

**格式二**

DIVIDE srcnum INTO srcnum GIVING {rsult [ROUNDED]} …

[ON SIZE ERROR stment]

[NOT ON SIZE ERROR stment2]

[END-DIVIDE]

**格式三**

DIVIDE srcnum BY srcnum GIVING {rsult [ROUNDED]} …

[ON SIZE ERROR stment]

[NOT ON SIZE ERROR stment2]

[END—DIVIDE]

**格式四**

DIVIDE srcnum INTO srcnum GIVING rsult [ROUNDED] REMAINDER remaind

[ON SIZE ERROR stment]

[NOT ON SIZE ERROR stment2]

[END—DIVIDE]

**格式五**

DIVIDE srcnum BY srcnum GIVING rsult [ROUNDED] REMAINDER remaind

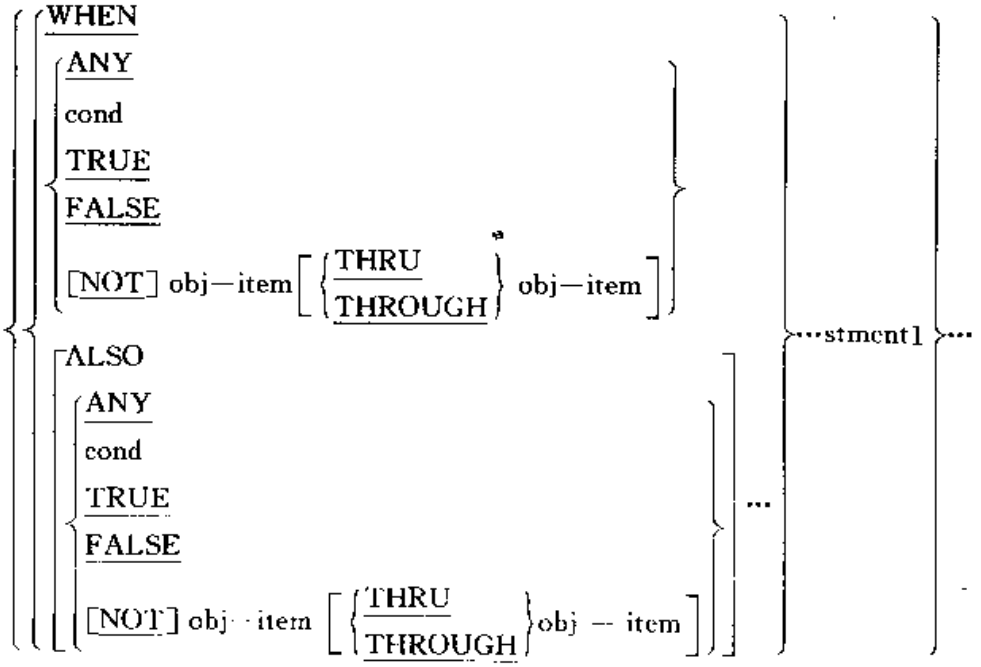
[ON SIZE ERROR stment]

[NOT ON SIZE ERROR stment2]

[END-DIVIDE]

**EVALUATE 语句**

$$\text{EVALUATE}\left\{\begin{matrix}\text{subj-item}\\ \text{TRUE}\\ \text{FALSE}\end{matrix}\right\}\left[\text{ALSO}\left\{\begin{matrix}\text{subj—item}\\ \text{TRUE}\\ \text{FALSE}\end{matrix}\right\}\right]\cdots$$




[ WHEN OTHER stment2]

[ END—EVALUATE]

### EXIT 语句

EXIT

### EXIT PROGRAM 语句

EXIT PROGRAM

### GO TO 语句

### 格式一

GO TO { proc—name}

### 格式二

GO TO { proc—name} …DEPENDING ON num

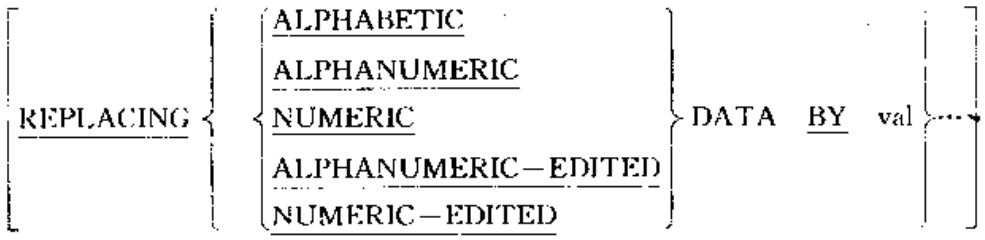
### IF 语句

IF condition THEN {{stment —1} … / NEXT SENTENCE}

[ELSE {stment—2} … [END—IF]
ELSE NEXT SENTENCE
END—IF]

### INITIALIZE 语句

INITIALIZE {fld—name} …

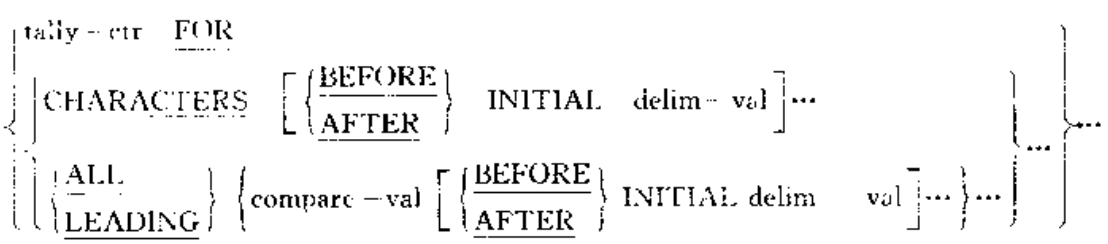



## INITIATE 语句

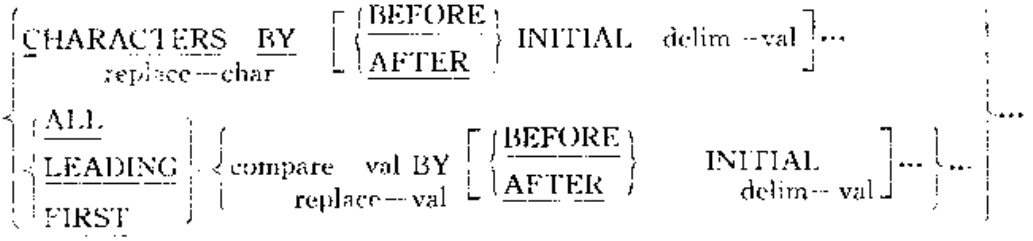
INITIATE {report-name} …

## INSPECT 语句

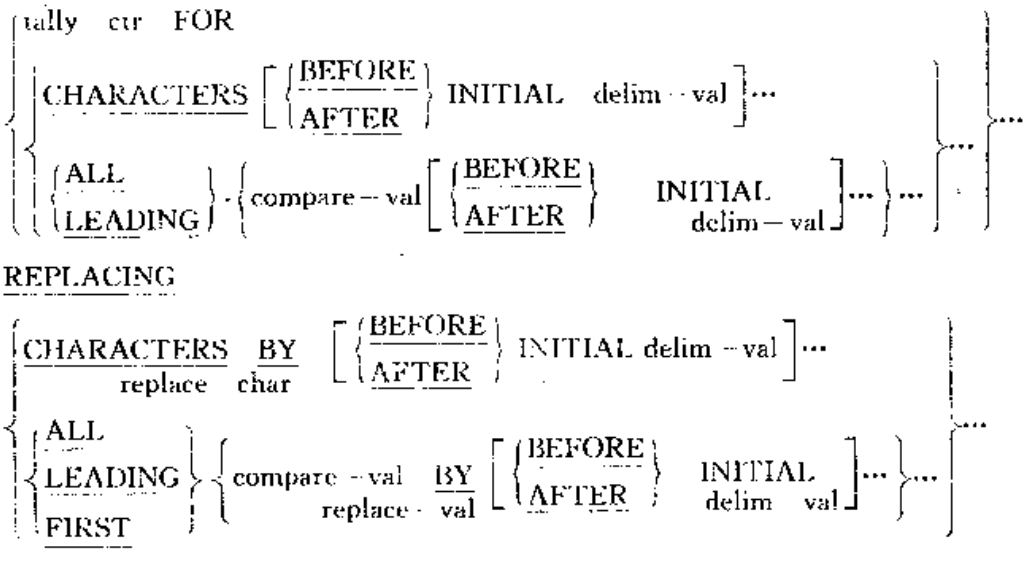
### 格式一




### 格式二




### 格式三



### 格式四

```
INSPECT src-string CONVERTING compare-chars TO convert chars
[{BEFORE|AFTER} INITIAL delim-val]…
```

## MERGE 语句

```
MERGE mergefile {ON {DESCENDING|ASCENDING} KEY {mergekey} …}…
    [COLLATING SEQUENCE IS alpha]
    USING infile {infile} …
{OUTPUT PROCEDURE IS first-proc[{THRU|THROUGH} end-proc]}
{GIVING {outfile} …}
```

## MOVE 语句

### 格式一

```
MOVE {src-item|lit} TO {dest-item} …
```

### 格式二

```
MOVE {CORRESPONDING|CORR} src-item TO dest-item
```

## MULTIPLY 语句

### 格式一

```
MULTIPLY srcnum BY {rsult [ROUNDED]} …
    [ON SIZE ERROR stment]
    [NOT ON SIZE ERROR stment2]
    [END-MULTIPLY]
```

### 格式二

```
MULTIPLY srcnum BY srcnum GIVING {rsult [ROUNDED]} …
    [ON SIZE ERROR stment]
    [NOT ON SIZE ERROR stment2]
    [END-MULTIPLY]
```

## OPEN 语句

**格式一——顺序、相对、索引文件**

```
OPEN
{ {INPUT  }
  {OUTPUT }
        file-name [ WITH  NO  REWIND] }...
  {EXTEND }
  {I-O    }  file-name ...
```

**格式二——报表生成文件**

```
OPEN {OUTPUT { file-name [ WITH  NO  REWIND ] } ...
     {EXTEND {file-name} ...                    }...
```

**PERFORM 语句**

**格式一**

```
PERFORM [first-proc [ {THRU    } end-proc ]]
        {THROUGH }
[ stment  END-PERFORM]
```

**格式二**

```
PERFORM [first-proc [ {THRU    } end-proc ]] repeat-count TIMES
        {THROUGH }
   [stment  END-PERFORM]
```

**格式三**

```
PERFORM [first-proc [ {THRU    } end-proc ]]
        {THROUGH }
                    [WITH  TEST {BEFORE} ] UNTIL  cond
                                {AFTER  }
   [ stment END-PERFORM]
```

**格式四**

```
PERFORM [first-proc [ {THRU    } end-proc ]]
        {THROUGH }
                    [WITH  TEST {BEFORE} ]
                                {AFTER  }
VARYING  var  FROM  init  BY  increm UNTIL  cond
[AFTER  var  FROM  init  BY  increm  UNTIL  cond] ...
[ stment  END-PERFORM]
```

**READ 语句**

**格式一**

READ file-name [NEXT] RECORD [INTO dest-item]
[AT END stment]
[NOT AT END stment2]
[END-READ]

**格式二**

READ file-name RECORD [INTO dest-item]
[KEY IS key-name]
[INVALID KEY stment]
[NOT INVALID KEY stment2]
[END-READ]

**RELEASE 语句**

RELEASE rec [FROM src-area]

**RETURN 语句**

RETURN smrg file RECORD [INTO dest-area]
AT END stment
[NOT AT END stment2]
[END-RETURN]

**REWRITE 语句**

REWRITE rec-name [FROM src-item]
[INVALID KEY stment]
[NOT INVALID KEY stment2]
[END-REWRITE]

**SEARCH 语句**

**格式一**

SEARCH src-table [VARYING pointr] [AT END stment]
{WHEN cond {stment / NEXT SENTENCE}}… [END-SEARCH]

**格式二**

SEARCH ALL src-table [AT END stment]

```
WHEN { elemnt { IS EQUAL TO } arg }
     { cond-name { IS =        }     }
[ AND { elemnt { IS EQUAL TO } arg } ]...
      { cond-name { IS =        }     }
{ stment      }
{ NEXT SENTENCE }   [END-SEARCH]
```

**SET 语句**

**格式一**

```
SET {rsult} ...TO val
```

**格式二**

```
SET {indx} ... { UP   BY } increm
               { DOWN BY }
```

**格式三**

```
SET {cond-name} ... TO TRUE
```

**格式四**

```
SET { { switch-name} ...TO { ON  } }...
                           { OFF }
```

**SORT 语句**

```
SORT sortfile { ON { DESCENDING } KEY {sortkey} ... }...
              { ASCENDING }
   [WITH DUPLICATES IN ORDER]
   [COLLATING SEQUENCE IS alpha]
{ INPUT  PROCEDURE IS first-proc [ { THRU    } end-proc ] }
{                                { THROUGH }            }
{ USING {infile} ...                                    }
{ OUTPUT PROCEDURE IS first-proc [ { THRU    } end-proc ] }
{                                { THROUGH }            }
{ GIVING {outfile} ...                                  }
```

**START 语句**

```
START file-name
```

```
[ KEY { IS EQUAL TO
      | IS =
      | IS GREATER THAN
      | IS >
      | IS NOT LESS THAN
      | IS NOT <
      | IS GREATER THAN OR EQUAL TO
      | IS > = } key-data ]
  [ INVALID KEY stment]
  [ NOT INVALID KEY stment2]
  [ END-START ]
```

### STOP 语句

```
STOP { RUN  }
     { disp }
```

### STRING 语句

```
STRING { {src string} …DELIMITED BY { delim } }…
       {                            { SIZE  } }
INTO dest string [WITH POINTER pointr]
  [ON OVERFLOW stment]
  [NOT ON OVERFLOW stment2]
  [END-STRING]
```

### SUBTRACT 语句

#### 格式一

```
SUBTRACT {num} …FROM {rsult [ROUNDED] } …
  [ ON SIZE ERROR stment ]
  [ NOT ON SIZE ERROR stment2]
  [ END-SUBTRACT ]
```

#### 格式二

```
SUBTRACT {num} …FROM num GIVING {rsult [ROUNDED] } …
  [ON SIZE ERROR stment]
  [NOT ON SIZE ERROR stment2]
  [END-SUBTRACT]
```

#### 格式三

```
SUBTRACT { CORRESPONDING } grp-1 FROM grp-2 [ROUNDED]
         { CORR          }
```

[ON SIZE ERROR stment]

[NOT ON SIZE ERROR stment2]

[END-SUBTRACT]

### SUPPRESS 语句

SUPPRESS PRINTING

### TERMINATE 语句

TERMINATE {report-name} …

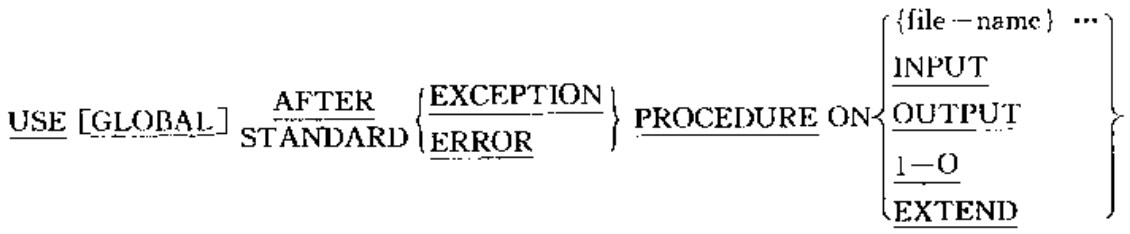
### UNSTRING 语句

UNSTRING src-string
[DELIMITED BY [ALL] delim [OR [ALL] delim] …]
INTO {dest string [ DELIMITER IN delim-dest]
[COUNT IN countr]} …

[WITH POINTER pointr]
[TALLYING IN tally-ctr]
[ON OVERFLOW stment]
[NOT ON OVERFLOW stment2]
[END-UNSTRING]

### USE 语句

#### 格式一

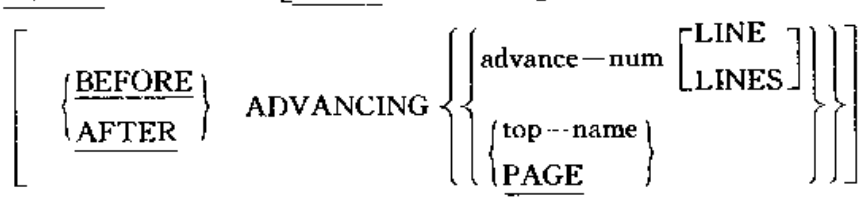


#### 格式二

USE [GLOBAL] BEFORE REPORTING group-data-name

### WRITE 语句

#### 格式一



[AT {END-OF-PAGE / EOP} stment]

[NOT AT {END-OF-PAGE / EOP} stment2]

[END-WRITE]

### 格式二

WRITE rec name [FROM src item]

[INVALID KEY stment]

[NOT INVALID KEY stment2]

[END WRITE]

## END PROGRAM 语句

END PROGRAM program-name.

## COPY 语句

### 格式

COPY text-name [ {OF / IN} library-name ]

[REPLACING

{ {==pseudo-text-1== / identifier-1 / literal-1 / *word-1* } BY { ==pseudo-text-2== / identifier-2 / literal-2 / *word-2* } }...

## REPLACE 语句

### 格式一

REPLACE {== pseudo-text-1== BY== pseudo-text-2==} ...

### 格式二

REPLACE OFF

## 其它语法格式

### Subscripting

{data-name / condition-name}

### Indexing

```
{data-name     } ( {index-name [ {+} literal-2 ]} ... )
{condition-name}   {literal-1                    }
```

### Reference Modification

```
data-name (leftmost-character-position : [ length ])
```

### Identifier

### 格式一

```
data-name [ qualification ] [ subscripting ] [ reference modification ]
```

### 格式二

```
data-name [ qualification ] [ indexing ] [ reference modification ]
```

### 关系条件

```
                                  Relational Operator                                  Object
                                  IS [NOT] GREATER THAN
                                  IS [NOT] >
                                  IS [NOT] LESS THAN
                                  IS [NOT] <
{identifier-1                }    IS [NOT] EQUAL TO    }   {identifier-2                }
{literal-1                   }    IS [NOT] =           }   {literal-2                   }
{arithmetic-expression-1     }    IS GREATER THAN OR EQUAL TO   {arithmetic expression 2}
                                  IS >=
                                  IS LESS THAN OR EQUAL TO
                                  IS <=
```

### 类别条件

```
                                  {NUMERIC          }
                                  {ALPHABETIC       }
identifier  IS  [NOT]             {ALPHABETIC-LOWER }
                                  {ALPHABETIC-UPPER }
                                  {class-name       }
```

### 符号条件

```
                                  {POSITIVE}
arithmetic-expression  IS  [NOT]  {NEGATIVE}
                                  {ZERO    }
```

### 取反简单条件

NOT simple—condition

### 条件与条件或

condition $\left\{ \left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\} \text{condition} \right\}\cdots$

### 复合缩写关系条件

relation—condition $\left\{ \left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\} [\underline{\text{NOT}}] [\text{relational operator}] \text{object} \right\}\cdots$

### ON SIZE ERROR Phrase

[NOT] ON SIZE ERROR stment

### INVALID KEY Phrase

[NOT] INVALID KEY stment

### AT END Phrase

[NOT] AT END stment

### FROM Phrase

record—name FROM identifier

### INTO Phrase

file—name INTO identifier

### Segmentation

section—name SECTION [segment number].

# 附录Ⅶ　常用字符与 ASCII 代码对照表

| ASCII 值 | 字符 | 控制字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | (null) | NUL | 032 | (space) | 064 | @ | 096 | ` | 128 | Ç | 160 | á | 192 | └ | 224 | α |
| 001 | ☺ | SOH | 033 | ! | 065 | A | 097 | a | 129 | ü | 161 | í | 193 | ┴ | 225 | β |
| 002 | ● | STX | 034 | " | 066 | B | 098 | b | 130 | é | 162 | ó | 194 | ┬ | 226 | Γ |
| 003 | ♥ | ETX | 035 | # | 067 | C | 099 | c | 131 | â | 163 | ú | 195 | ├ | 227 | π |
| 004 | ♦ | EOT | 036 | $ | 068 | D | 100 | d | 132 | ä | 164 | ñ | 196 | ─ | 228 | Σ |
| 005 | ♣ | END | 037 | % | 069 | E | 101 | e | 133 | à | 165 | Ñ | 197 | ┼ | 229 | σ |
| 006 | ♠ | ACK | 038 | & | 070 | F | 102 | f | 134 | å | 166 | ª | 198 | ╞ | 230 | μ |
| 007 | (beep) | BEL | 039 | ' | 071 | G | 103 | g | 135 | ç | 167 | º | 199 | ╟ | 231 | τ |
| 008 | ■ | BS | 040 | ( | 072 | H | 104 | h | 136 | ê | 168 | ¿ | 200 | ╚ | 232 | Φ |
| 009 | (tab) | HT | 041 | ) | 073 | I | 105 | i | 137 | ë | 169 | ⌐ | 201 | ╔ | 233 | θ |
| 010 | (line feed) | LF | 042 | * | 074 | J | 106 | j | 138 | è | 170 | ¬ | 202 | ╩ | 234 | Ω |
| 011 | (home) | VT | 043 | + | 075 | K | 107 | k | 139 | ï | 171 | 1/2 | 203 | ╦ | 235 | δ |
| 012 | (form feed) | FF | 044 | , | 076 | L | 108 | l | 140 | î | 172 | 1/4 | 204 | ╠ | 236 | ∞ |
| 013 | (carriage return) | CR | 045 | - | 077 | M | 109 | m | 141 | ì | 173 | ¡ | 205 | ═ | 237 | φ |
| 014 | ♫ | SO | 046 | . | 078 | N | 110 | n | 142 | Ä | 174 | « | 206 | ╬ | 238 | ∈ |
| 015 | ☼ | SI | 047 | / | 079 | O | 111 | o | 143 | Å | 175 | » | 207 | ╧ | 239 | ∩ |
| 016 | ► | DLE | 048 | 0 | 080 | P | 112 | p | 144 | É | 176 | ▒ | 208 | ╨ | 240 | ≡ |
| 017 | ◄ | DC1 | 049 | 1 | 081 | Q | 113 | q | 145 | æ | 177 | ▓ | 209 | ╤ | 241 | ± |
| 018 | ↕ | DC2 | 050 | 2 | 082 | R | 114 | r | 146 | Æ | 178 | ▓ | 210 | ╥ | 242 | ≥ |
| 019 | ‼ | DC3 | 051 | 3 | 083 | S | 115 | s | 147 | ô | 179 | ▒ | 211 | ╙ | 243 | ≤ |
| 020 | ¶ | DC4 | 052 | 4 | 084 | T | 116 | t | 148 | ö | 180 | ┤ | 212 | ╘ | 244 | ⌠ |
| 021 | § | NAK | 053 | 5 | 085 | U | 117 | u | 149 | ò | 181 | ╡ | 213 | ╒ | 245 | ⌡ |
| 022 | ▬ | SYN | 054 | 6 | 086 | V | 118 | v | 150 | û | 182 | ╢ | 214 | ╓ | 246 | ÷ |
| 023 | ↨ | ETB | 055 | 7 | 087 | W | 119 | w | 151 | ù | 183 | ╖ | 215 | ╫ | 247 | ≈ |
| 024 | ↑ | CAN | 056 | 8 | 088 | X | 120 | x | 152 | ÿ | 184 | ╕ | 216 | ╪ | 248 | ° |
| 025 | ↓ | EM | 057 | 9 | 089 | Y | 121 | y | 153 | Ö | 185 | ╣ | 217 | ┘ | 249 | ∙ |
| 026 | → | SUB | 058 | : | 090 | Z | 122 | z | 154 | Ü | 186 | ║ | 218 | ┌ | 250 | · |
| 027 | ← | ESC | 059 | ; | 091 | [ | 123 | { | 155 | ¢ | 187 | ╗ | 219 | █ | 251 | √ |
| 028 | ∟ | FS | 060 | < | 092 | \ | 124 | \| | 156 | £ | 188 | ╝ | 220 | ▄ | 252 | n |
| 029 | ◆ | GS | 061 | = | 093 | ] | 125 | } | 157 | ¥ | 189 | ╜ | 221 | ▌ | 253 | 2 |
| 030 | ▲ | RS | 062 | > | 094 | ^ | 126 | ~ | 158 | Pt | 190 | ╛ | 222 | ▐ | 254 | ■ |
| 031 | ▼ | US | 063 | ? | 095 | _ | 127 | ⌂ | 159 | ƒ | 191 | ┐ | 223 | ▀ | 255(blank 'FF') | |

注：128～255 是 IBM PC（长城 0520）上专用的。表中 00～127 是标准的。

# 附录Ⅷ　常用字符与EBCPIC代码对照表

| 字　符 | 十进制 | 十六进制 | 八进制 | 二进制 |
|---|---|---|---|---|
| . | 75 | 4B | 113 | 0100 1011 |
| < | 76 | 4C | 114 | 0100 1100 |
| ( | 77 | 4D | 115 | 0100 1101 |
| + | 78 | 4E | 116 | 0100 1110 |
| \| | 79 | 4F | 117 | 0100 1111 |
| & | 80 | 50 | 120 | 0101 0000 |
| ! | 90 | 5A | 132 | 0101 1010 |
| $ | 91 | 5B | 133 | 0101 1011 |
| * | 92 | 5C | 134 | 0101 1100 |
| ) | 93 | 5D | 135 | 0101 1101 |
| ; | 94 | 5E | 136 | 0101 1110 |
| ¬ | 95 | 5F | 137 | 0101 1111 |
| - | 96 | 60 | 140 | 0110 0000 |
| / | 97 | 61 | 141 | 0110 0001 |
| \| | 106 | 6A | 152 | 0110 1010 |
| , | 107 | 6B | 153 | 0110 1011 |
| % | 108 | 6C | 154 | 0110 1100 |
| — | 109 | 6D | 155 | 0110 1101 |
| > | 110 | 6E | 156 | 0110 1110 |
| ? | 111 | 6F | 157 | 0110 1111 |
| : | 122 | 7A | 172 | 0111 1010 |
| # | 123 | 7B | 173 | 0111 1011 |
| @ | 124 | 7C | 174 | 0111 1100 |
| ' | 125 | 7D | 175 | 0111 1101 |
| = | 126 | 7E | 176 | 0111 1110 |
| " | 127 | 7F | 177 | 0111 1111 |
| a | 129 | 81 | 201 | 1000 0001 |
| b | 130 | 82 | 202 | 1000 0010 |
| c | 131 | 83 | 203 | 1000 0011 |
| d | 132 | 84 | 204 | 1000 0100 |
| e | 133 | 85 | 205 | 1000 0101 |
| f | 134 | 86 | 206 | 1000 0110 |
| g | 135 | 87 | 207 | 1000 0111 |
| h | 136 | 88 | 210 | 1000 1000 |
| i | 137 | 89 | 211 | 1000 1001 |
| j | 145 | 91 | 221 | 1001 0001 |
| k | 146 | 92 | 222 | 1001 0010 |

续表

| 字　符 | 十进制 | 十六进制 | 八进制 | 二进制 |
|---|---|---|---|---|
| l | 147 | 93 | 223 | 1001 0011 |
| m | 148 | 94 | 224 | 1001 0100 |
| n | 149 | 95 | 225 | 1001 0101 |
| o | 150 | 96 | 226 | 1001 0110 |
| p | 151 | 97 | 227 | 1001 0111 |
| q | 152 | 98 | 230 | 1001 1000 |
| r | 153 | 99 | 231 | 1001 1001 |
| s | 162 | A2 | 242 | 1010 0010 |
| t | 163 | A3 | 243 | 1010 0011 |
| u | 164 | A4 | 244 | 1010 0100 |
| v | 165 | A5 | 245 | 1010 0101 |
| w | 166 | A6 | 246 | 1010 0110 |
| x | 167 | A7 | 247 | 1010 0111 |
| y | 168 | A8 | 250 | 1010 1000 |
| z | 169 | A9 | 251 | 1010 1001 |
| ~ | 161 | A1 | 241 | 1010 0001 |
| { | 192 | C0 | 300 | 1100 0000 |
| } | 208 | D0 | 320 | 1101 0000 |
| A | 193 | C1 | 301 | 1100 0001 |
| B | 194 | C2 | 302 | 1100 0010 |
| C | 195 | C3 | 303 | 1100 0011 |
| D | 196 | C4 | 304 | 1100 0100 |
| E | 197 | C5 | 305 | 1100 0101 |
| F | 198 | C6 | 306 | 1100 0110 |
| G | 199 | C7 | 307 | 1100 0111 |
| H | 200 | C8 | 310 | 1100 1000 |
| I | 201 | C9 | 311 | 1100 1001 |
| J | 209 | D1 | 321 | 1101 0001 |
| K | 210 | D2 | 322 | 1101 0010 |
| L | 211 | D3 | 323 | 1101 0011 |
| M | 212 | D4 | 324 | 1101 0100 |
| N | 213 | D5 | 325 | 1101 0101 |
| O | 214 | D6 | 326 | 1101 0110 |
| P | 215 | D7 | 327 | 1101 0111 |
| Q | 216 | D8 | 330 | 1101 1000 |
| R | 217 | D9 | 331 | 1101 1001 |
| S | 226 | E2 | 342 | 1110 0010 |
| T | 227 | E3 | 343 | 1110 0011 |
| U | 228 | E4 | 344 | 1110 0100 |
| V | 229 | E5 | 345 | 1110 0101 |

续表

| 字　符 | 十进制 | 十六进制 | 八进制 | 二进制 |
|---|---|---|---|---|
| W | 230 | E6 | 346 | 1110 0110 |
| X | 231 | E7 | 347 | 1110 0111 |
| Y | 232 | E8 | 350 | 1110 1000 |
| Z | 233 | E9 | 351 | 1110 1001 |
| \ | 221 | E0 | 340 | 1110 0000 |
| 0 | 240 | F0 | 360 | 1111 0000 |
| 1 | 241 | F1 | 361 | 1111 0001 |
| 2 | 242 | F2 | 362 | 1111 0010 |
| 3 | 243 | F3 | 363 | 1111 0011 |
| 4 | 244 | F4 | 364 | 1111 0100 |
| 5 | 245 | F5 | 365 | 1111 0101 |
| 6 | 246 | F6 | 366 | 1111 0110 |
| 7 | 247 | F7 | 367 | 1111 0111 |
| 8 | 248 | F8 | 368 | 1111 1000 |
| 9 | 249 | F9 | 369 | 1111 1001 |
| 换行(LF) | 37 | 25 | 045 | 0010 0101 |
| 空格(SP) | 64 | 40 | 100 | 0100 0000 |
| 回车(CR) | 13 | OD | 015 | 0000 1101 |